

Softwaretechnik SS 2004

Dokumentation

Eine Gruppenarbeit von:

Daniel Schneidereit
Alexander Roth
Jörg Schweitzer
Mehrdad Mojarabi-Tabrizi

„Fang den Hut“-Dokumentation - Inhaltsverzeichnis

Einleitung

Die vorliegende Dokumentation beschreibt die Vorgehensweise und die einzelnen Schritte bei der Konzeption, Design und Implementierung des Spiels „Fang den Hut“. Des weiteren enthält diese Dokumentation auch das detaillierte Fachkonzept, Systemdokumentation und Benutzerhandbuch zu dem Spiel.

Projektziele

Das Projekt hat zum Ziel die Erstellung des Spiels „Fang den Hut“ in Visual C++ Version 6. Die Anwendung soll als eine SDI-Anwendung unter Verwendung des Microsoft Foundation Classes Frameworks realisiert werden. Als wichtigstes Kriterium bei der Konzeption der Anwendung gilt die Umsetzung der „Model-View-Controller Archtektur“. Die einzelnen Projektphasen bestehen aus Analyse, Design, Implementierung und Dokumentationserstellung

Für die Analyse und das Design des Projektes, als auch für die Erstellung von UML-Diagrammen wird das Case-Tool "Together Control Center 6.1." eingesetzt. Das Spiel ist von einer Projektgruppe, bestehend aus vier Personen in einem Zeitraum von ca. 14 Wochen zu erstellen.

Fachliche Anforderungen

- grundsätlich entsprechen die fachlichen Anforderungen den Spielregeln (siehe Benutzerhandbuch)
- die Anzahl der Spieler beträgt 2 bis 4 Personen
- Anmeldung der Spieler und Auswahl einer Farbe am Anfang des Spiels
- alle Teilnehmer können wechselseitig an einem Bildschirm miteinander spielen
- jeder Teilnehmer kommt nacheinander dran und würfelt
- Darstellung der möglichen Züge nach dem Würfeln und Auswählen des gewünschten Hutes
- der ausgewählte Hut wird entsprechend der Feldauswahl bewegt
- je nach dem auf welchem Feld der Hut landet, können fremde Hüte gefangengenommen werden.
- wenn das Spiel noch nicht beendet werden kann, so darf der nächste Spieler würfeln. Außer der aktive Spieler hat zuletzt eine 6 gewürfelt, so kommt er erneut dran
- Bedienung der Spiels ausschließlich mit einer Computermaus. Nur zur Eingabe der Spielernamen ist Tastaturbedienung notwendig
- Speicherung und Wiederherstellung der verschiedenen Spielstände
- Spielen nach Standard- und Alternativspielregeln

Implementierungsanforderungen

- Projektprogrammierung in Visual C++ Version 6
- Umsetzung der Model-View-Controller Architektur
- Verwendung des Microsoft Foundation Classes Frameworks
- Umsetzung der Anwendung als eine SDI-Anwendung
- Bedienung des Spiels ausschließlich mit einer Computermaus durch Drag-and-Drop und Select-and-Click
- Spielstandspeicherung und –wiederherstellung mit Hilfe der Serialisierung aus dem MFC-Framework

Benutzerdokumentation

Installationsanleitung

- Erstellen sie an einer beliebigen Stelle auf ihrer Festplatte einen Ordner für das Spiel z.B C:\Spiele\FangDenHut
- Kopieren Sie den Inhalt des Ordners [CDROM]:\FangDenHut\setup in den von ihnen angelegten Ordner
- Zum Spielen des Spiels führen sie einen Doppel-Klick auf die Datei FangDenHut.exe aus

Spielregeln

<u>Grundregel</u>

Zu Beginn wählt jeder Spieler sechs Hütchen einer Farbe und setzt sie auf das runde Feld (das „Versteck“) derselben Farbe. Von hier aus starten die Hütchen und nach hier bringen die Spieler bzw. die Hütchen ihre Gefangenen wieder zurück. Im Versteck, das man nach dem Verlassen nur wieder mit Gefangenen betreten darf, ist man unangreifbar und kann nicht gefangen werden. Ebenso geschützt sind die Hüte auf den Ruhebänkchen (dunklen Feldern) des Spielplanes.

Die Spieler würfeln nacheinander der Reihe nach. Die Hütchen werden immer soviele Felder weitergerückt, als Augen auf dem Würfel sind. Die Richtung ist nicht vorgeschrieben - jeder Spieler kann seine Hütchen nach allen Seiten (vor- und rückwärts) bewegen, in einem Zug jedoch immer nur in einer Richtung (Ausnahme siehe weiter unten). Es steht den Spielern frei, welche Hütchen sie bewegen wollen, doch dürfen sie selbstverständlich bei jedem Wurf nur mit einem Hütchen ziehen.

Den Spielern bleibt es überlassen, gleich anfangs mit allen, oder nach und nach mit je einem Hütchen in das Spielfeld einzurücken. Ziel des Spieles ist es, die Hütchen der anderen Farben zu fangen und zurück in das eigene Versteck zu bringen. Gefangen ist ein Hütchen, wenn ein anderes Hütchen direkt auf dasselbe Feld kommt. Man stülpt dann einfach sein eigenes Hütchen über das gefangene.

Auf einem Feld können ohne weiteres mehrere Hütchen der gleichen Farbe stehen. Trifft aber ein Hütchen einer anderen Farbe auf dieses Feld, so kann es alle dort stehenden Hütchen gefangennehmen. Ein Spieler, der mit einem oder mehreren gefangenen Hütchen unterwegs ist, kann noch weitere Hütchen einfangen, wenn er durch einen entsprechenden Wurf ein bereits besetztes Feld erreicht.

Gefangene Hütchen, die nicht auf einem Ruhebänkchen stehen, oder in einem Versteck sichergestellt sind, können wieder befreit werden. Dieses Abjagen ist dann möglich, wenn der beraubte Spieler seinem Gegner nacheilt und diesen einzuholen versucht. Gelingt ihm dies, indem er auf das gleiche Feld gelangt, so setzt er sein Hütchen auf das des Gegners samt allen darunter befindlichen Hütchen. Dann muß er versuchen, mit seiner Beute schleunigst in sein Versteck zu gelangen. Dort kann er sein bisher gefangenes Hütchen befreien und wieder verwenden, während alle übrigen Gefangenen sichergestellt werden.

Um gefangene Hütchen in sein Versteck zu schaffen, ist es nicht erforderlich, genau die Zahl zu würfeln, mit der das Versteck erreicht werden kann. Beispiel: Man steht 2 Felder vor dem Versteck und wirft 5 Augen. Nun bewegt man den Hut in das Versteck (das als ein Feld mitgezählt werden muß) und setzt dort Gefangenen ab. Mit den noch übrigen Punkten darf man nun wieder ein Hütchen vom Versteck auf das Spielfeld setzen. Dies ist der einzige Fall, in dem man in einem Zug die Richtung umdrehen darf.

Gewinner ist, wer zuletzt noch eigene Hütchen auf dem Spielfeld hat, wenn alle Hütchen der Gegner gefangen sind. Die Zahl der gefangenen Hütchen gibt dabei nicht den Ausschlag.

<u>Alternative Regel</u>

Man spielt wie bei der Grundregel. Alle Hütchen aber, die aus dem Versteck auf den Spielplan gebracht werden, dürfen nicht mehr in das Versteck zurück. Dadurch ist es auch nicht möglich, gefangene Hütchen sicherzustellen. So bleibt zum Schluß nur ein großer Turm übrig, dessen Besitzer das Spiel gewinnt.

Bedienungsanleitung

Ein neues Spiel starten

Starten Sie das Fang den Hut Programm. (Unter Windows mit einem Doppelklick auf "FangDenHut.exe".)

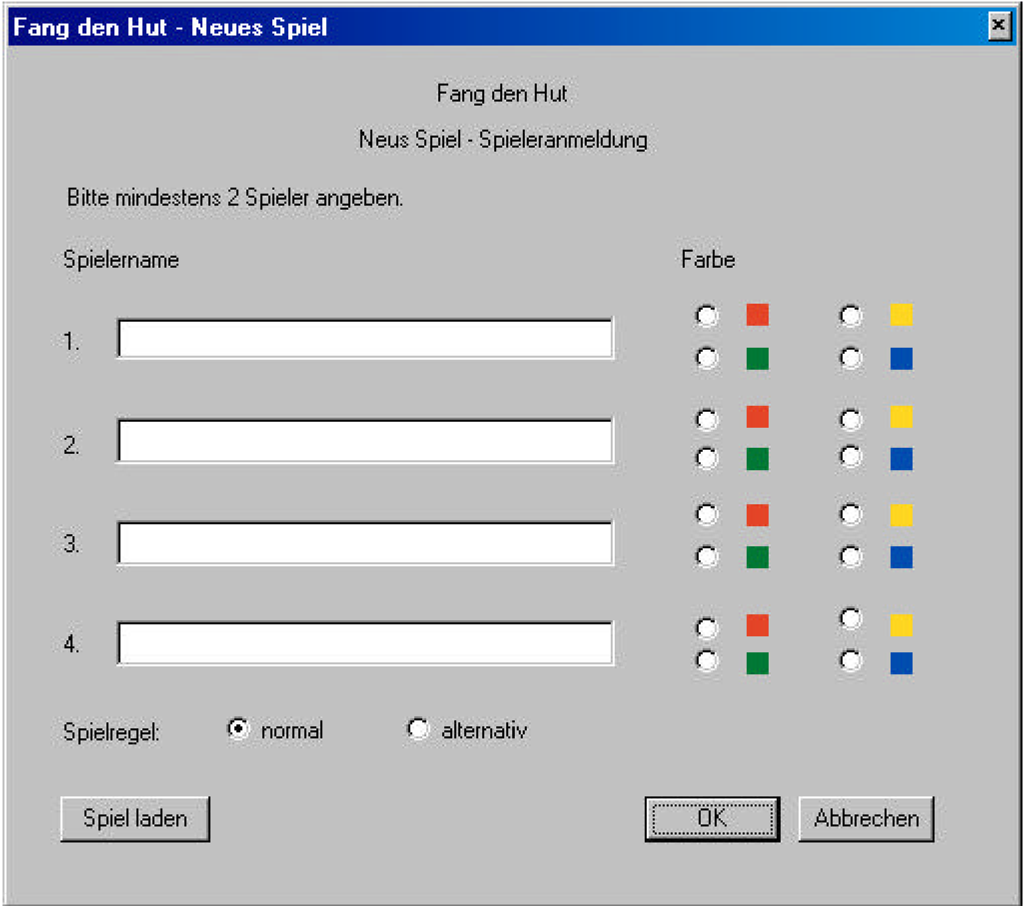


Abbildung 1: Anmeldebildschirm

Sie sehen dann den Anmeldebildschirm (Abb.: 1) vor sich. Hier nehmen Sie die nötigen Einstellungen für ein neues Spiel vor. Da man Fang den Hut nicht allein spielen kann müssen Sie 2, 3 oder 4 Spieler anlegen. Dazu geben Sie einfach die Namen der Spieler in die weißen Felder ein. Neben dem Namen eines Spielers können sie seine Farbe auswählen. Klicken sie dazu einfach auf den weißen Kreis neben der gewünschten Farbe.

Jede Farbe kann nur einmal vergeben werden, achten Sie also darauf, dass jeder Spieler eine andere Farbe erhält. Unter den Spielernamen haben Sie die Möglichkeit zwischen den normalen und den alternativen Regeln zu entscheiden. (Alternative Regeln werden in der Fang den Hut Regelbeschreibung erklärt)

Wenn Sie alle Ihre Eingaben getätigt haben bestätigen sie mit "OK". Sollte alles richtig eingestellt sein, kommen Sie nun zum Fang den Hut Spielfeld, und können loslegen. Haben Sie bei der Eingabe einen Fehler gemacht, bekommen Sie eine Fehlermeldung. Kontrollieren Sie ihre Eingaben und versuchen Sie es erneut. (Häufigste Fehler: nur ein Spieler ausgewählt, Farbe mehrfach vergeben)

Fang den Hut spielen

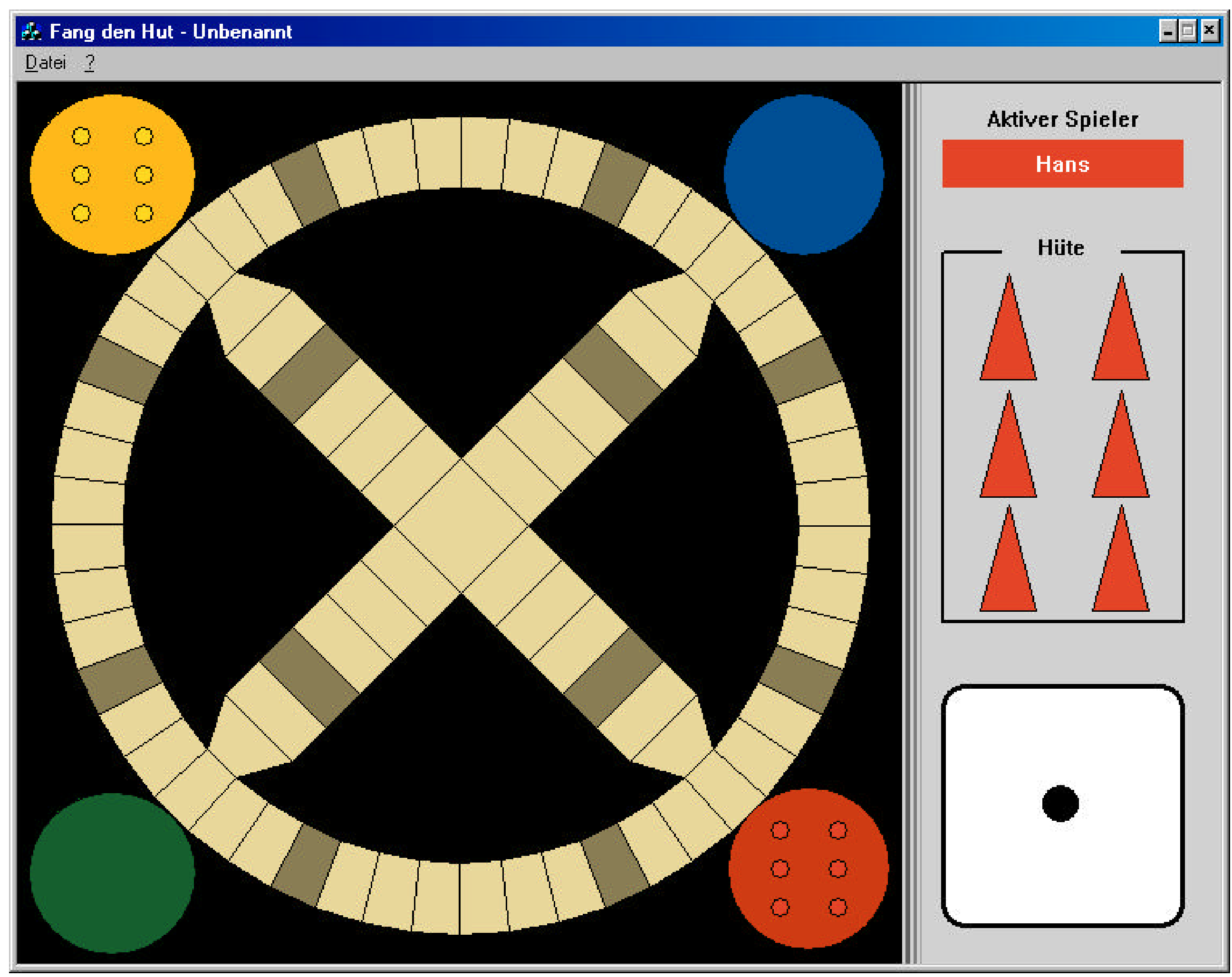


Abbildung 2: Spielbrett

Nachdem Sie ihr Spiel eingerichtet haben, sehen Sie das Fang den Hut Spielfeld vor sich (Abb.: 2). Das Spielfeld ist in zwei Bereiche unterteilt. Zum Einen das eigentliche Spielbrett (im linken Bereich), zum Anderen zusätzliche Informationen wie Spielername, Farbe und gewürfelte Augenzahl (im rechten Bereich). Unter "Aktiver Spieler" sehen sie den Namen des Spielers, der gerade am Zug ist. Der Name ist mit der Farbe des Spielers hinterlegt. Das Spielbrett, besteht aus den vier Heimfeldern, den Spielfeldern, und den Hüten. Die Sicherheitsfelder sind abgedunkelt dargestellt, die Hüte werden durch farbige Kreise auf den Feldern repräsentiert.

Wenn Sie nun die Maus über einen Hut bewegen, wird dieser im Fenster "Hüte" unter dem Spielernamen, angezeigt. Dort sehen Sie auch, ob mehrere Hüte einer Farbe auf einem Feld stehen, und welche Gefangene die Hüte haben. Der Würfel funktioniert völlig automatisch und zeigt an, wie viele Schritte Sie mit einem Hut ihrer Farbe in dieser Runde laufen müssen.

Zu Beginn des Spiels stehen alle Hüte auf den Heimfeldern. Wenn Sie die Maus auf einen Hut ihrer Farbe bewegen, sehen Sie also beim ersten Zug 6 identische Hüte im Auswahlfenster. Durch Klicken auf den Hut wählen Sie ihn aus. Sollten mehrere Hüte auf einem Feld stehen, können Sie mehrfach auf den Hut klicken, und so die zur Verfügung stehenden Hüte durchgehen. Der ausgewählte Hut wird im Auswahlfenster mit einem roten Rahmen markiert. Wenn Sie einen Hut ausgewählt haben, können sie die Felder auf die er sich bewegen kann, daran erkennen, dass sie hellblau gefärbt werden. Die Sicherheitsfelder werden etwas dunkler gefärbt. Um den Hut zu bewegen, klicken sie einfach auf das gewünschte Zielfeld. Ihr Hut wird dann automatisch auf das neue Feld bewegt, und der nächste Spieler ist an der Reihe. Sollten Sie sich dazu entscheiden lieber einen anderen Ihrer Hüte zu bewegen können Sie den aktuellen mittels einem rechten Mausklick wieder abwählen. Außerdem ist es möglich einfach einen anderen Ihrer Hüte anzuklicken. Die gültigen Spielfelder für diesen Hut werden dann sofort angezeigt.

Besondere Spielzüge

Doppelt ziehen bei "6"

Wenn Sie eine 6 gewürfelt haben, bewegen Sie den Hut wie gewohnt. Danach sind Sie ein weiteres Mal dran. Beachten Sie, dass sich die Augenzahl auf dem Würfel verändert hat. (Es sei denn Sie haben so viel Glück, noch eine 6 zu würfeln.) Machen Sie auch Ihren nächsten Zug wie gewohnt.

Auf ein besetztes Feld ziehen

Wenn sie auf ein Feld ziehen, auf dem sich andere Hüte befinden, gibt es grundsätzlich zwei Möglichkeiten:

1) Die Hüte auf dem Zielfeld gehören Ihnen.

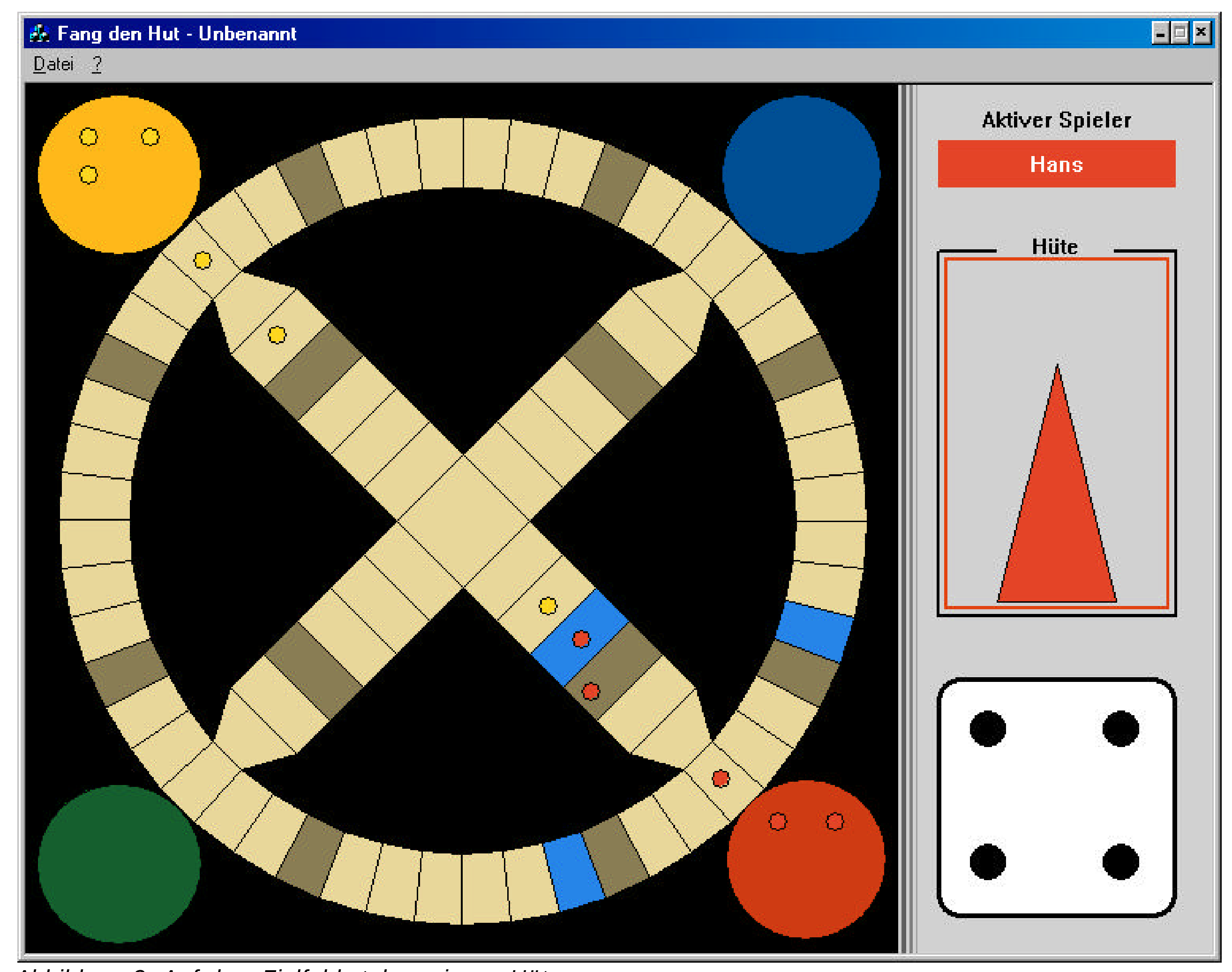


Abbildung 3: Auf dem Zielfeld stehen eigene Hüte

In diesem Fall, werden alle Hüte, die sich auf einem Feld befinden nur durch einen Kreis mit der entsprechenden Farbe dargestellt, um eine größere Übersicht zu gewährleisten. Die Hüte werden in dem Auswahlfenster genauer dargestellt.

2) Auf dem Zielfeld stehen fremde Hüte

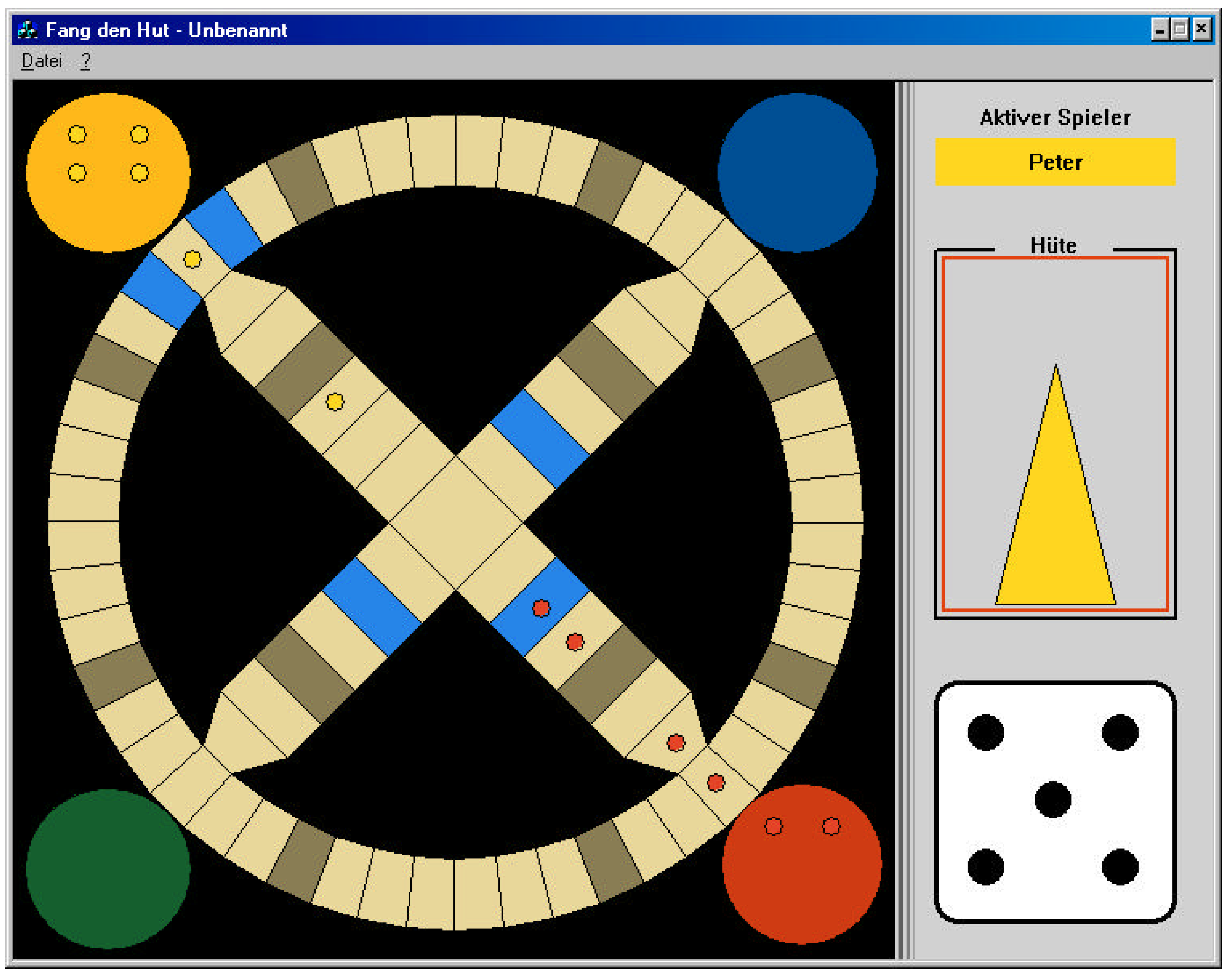


Abbildung 4: Fremde Hüte auf dem Zielfeld

Die fremden Hüte werden von Ihrem Hut gefangen genommen. Sie können das daran erkennen, dass im Auswahlfenster, unter ihrem Hut, der eben Gefangene und seine bereits Gefangenen dargestellt werden.

Sie sollten auf diese Darstellung achten, damit sie richtig abwägen können, welchen Hut Sie bewegen wollen.

Sonderfall: Auf einem Sicherheitsfeld können sie keine Hüte fangen, ihr Hut steht dann einfach nur neben den anderen fremden Hüten auf diesem Feld.

Gefangene Hüte "abliefern"

In den normalen Spielregeln besteht die Möglichkeit, das ein Hut seine Gefangenen zu Hause abliefert. Diese werden dann aus dem Spiel genommen. Sollten sich unter den Gefangenen einige Ihrer eigenen befinden, werden diese befreit und auf Ihr Heimatfeld plaziert. Ein Hut kann nur nach Hause zurück, wenn er Gefangene besitzt. Sollten Sie eine genügend Hohe Augenzahl gewürfelt haben um Ihr Heimatfeld zu erreichen, wird dieses wie alle anderen gültigen Felder mit markiert. Die einzige Besonderheit besteht nun darin, dass übrige Augen nun ebenfalls verbraucht werden müssen. Es werden Ihnen also sofort neue Zielfelder angezeigt. Ein Hutwechsel ist in dieser Situation nicht möglich. Bitte beachten Sie, das dieser Zugzwang natürlich nicht besteht, wenn sie Ihr Heimatfeld mit einer genau passenden Augenzahl erreichen.

Bei den alternativen Spielregeln entfällt dieser Spielzug.

Spiel speichern und laden

Sie können eine Partie Fang den Hut unterbrechen und zu einem späteren Zeitpunkt fortsetzen.

Klicken in der Menüleiste auf "Datei" und dann auf "Speichern".

Jetzt können sie einen Namen für ihren Spielstand (Savegame) angeben. Bestätigen sie mit "Speichern" und ihr Spielstand wird gespeichert.

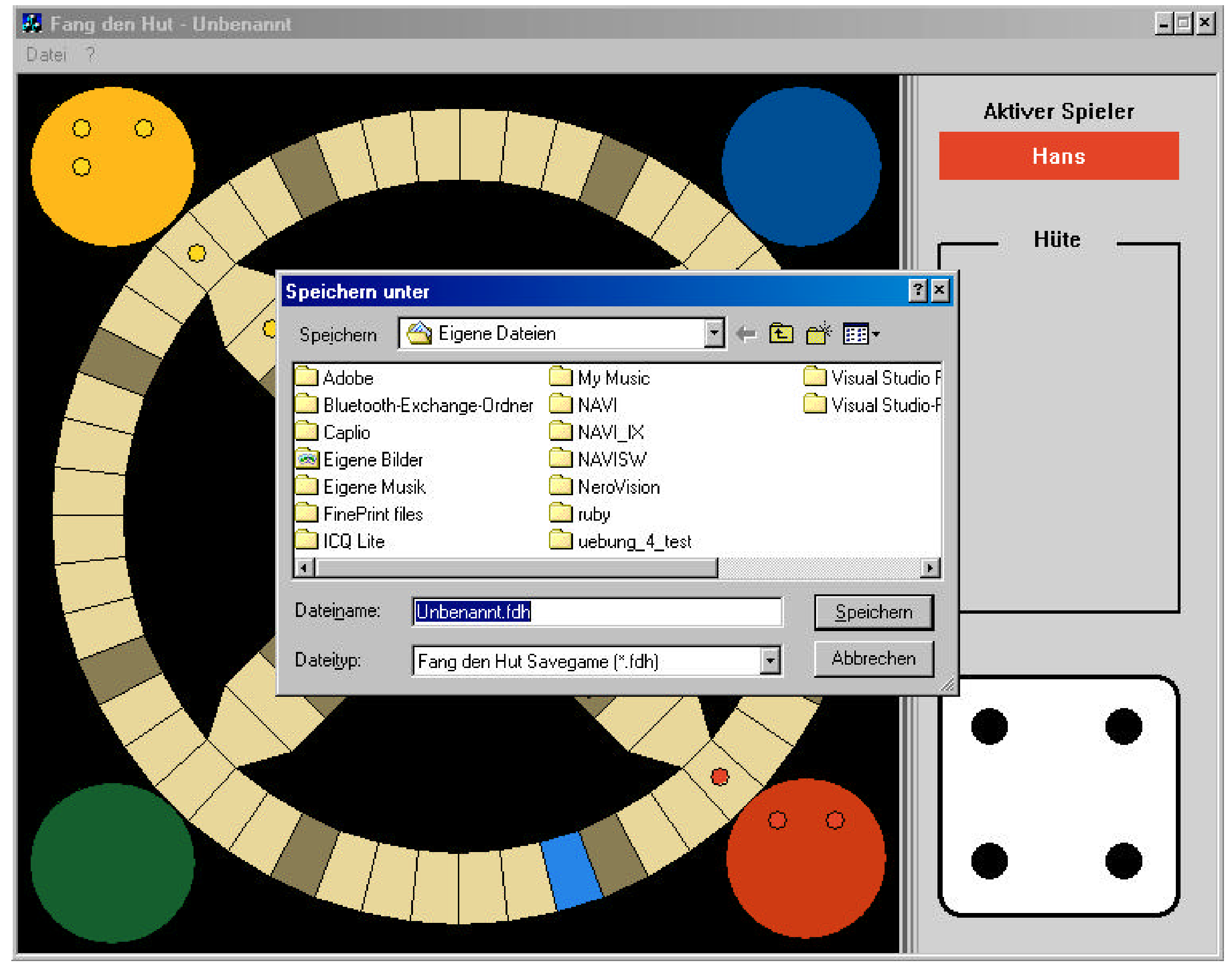


Abbildung 5: Spiel speichern

Wenn sie den aktuell gespeicherten Spielstand überschreiben wollen, können sie wieder auf "Speichern" klicken. Um einen neuen Spielstand zu speichern wählen sie bitte "Datei" -> "Speichern unter...".

Den Spielstand können sie zu Beginn des Spiels, oder mitten während eines laufenden Spiels laden. Dazu klicken sie im Einstellungsbildschirm am Anfang auf "Spiel laden" und wählen Ihren zuvor gespeicherten Spielstand aus. Während des Spiels können Sie mit "Datei" -> "Öffnen" ebenfalls einen Spielstand laden.

Spielende

Wenn es Ihnen gelungen ist alle Hüte der anderen Spieler gefangen zu nehmen, haben Sie das Spiel gewonnen.

Herzlichen Glückwunsch.

Sie können jetzt entweder zum Einstellungsbildschirm zurückgehen und ein neues Spiel starten oder Fang den Hut beenden.

Spielabbruch

Sie können das Spiel jederzeit mit "Datei" -> "Beenden" abbrechen.

Das Programm wird Sie zur Sicherheit noch fragen, ob Sie ihren Spielstand sichern möchten.

Systemdokumentation

Anforderungsdefinition

- Projektprogrammierung in Visual C++ Version 6
- Umsetzung der Model-View-Controller Architektur
- Verwendung des Microsoft Foundation Classes Frameworks
- Umsetzung der Anwendung als eine SDI-Anwendung
- Bedienung des Spiels ausschließlich mit einer Computermaus durch Drag-and-Drop und Select-and-Click
- Spielstandspeicherung und –wiederherstellung mit Hilfe der Serialisierung aus dem MFC-Framework

MVC-Architektur und Architekturentscheidungen

Die Anwendung „Fang den Hut“ wurde nach der Model-View-Controller Architektur erstellt. Diese unterstützt eine bessere Trennung (Entkopplung) der einzelnen Anwendungsbereiche Datenhaltung, Anwendungslogik und Präsentation. Sie dient auch einer einfacheren Pflege und Wartung, Systemerweiterung/-anpassung und Wiederverwendbarkeit. Somit sollte ein späterer Austausch von zum Beispiel der Präsentationsschicht einfacher und ohne Einflußnahme auf andere Anwendungsbestandteile erfolgen.

Um die einzelnen Komponentenbereiche Model, View und Controller noch mehr von einander zu entkoppeln wurde eine zusätzliche Schnittstellen-Klasse (`CInterface`) nach dem Fasaden-Muster eingeführt. Sie ist für die Kommunikation bzw. für die Nachrichtenweiterleitung zwischen den Anwendungsbestandteilen zuständig.

Beim Entwurf des Klassenmodels wurden auch folgende Entwurfsmuster und Konstrukte eingesetzt:0

- Vererbung
- Spezialisierung
- Expert
- Don‘ Talk to Strangers
- Serialisierung

Da die Behandlung des State-Patterns in der Vorlesung Softwaretechnik ziemlich am Ende der Lehrveranstaltung statt fand, konnte dieser beim Design nicht mehr berücksichtigt werden.

Zum Anwendungsbereich View gehören Klassen welche die Funktionalität zur Darstellung des Spielbrettes, der Spielfiguren, des Würfels und der Spielerinformationen enthalten. Sie besitzen keine Anwendungslogik und sind nur für die Darstellung zuständig.

Der Controller besitzt die Zuständigkeiten eines „Spielleiters“, denn im Controller sind die Spielregeln abgebildet. Er entscheidet während des Spielverlaufs abhängig vom Spielzustand und den auftretenden Ereignissen welche Aktion als nächste ausgeführt werden soll bzw. darf.

In dem Anwendungsbereich Model werden die Zustandsinformationen über das Spielbrett, den Würfel, die Spieler und ihre Hüte festgehalten.

UseCases

- Simple (siehe Anhang)
- Complex (siehe Anhang)

Klassenmodel

- Diagramm (siehe Anhang)
- Klassenbeschreibungen (siehe Anhang)

Sequenzdiagramme

- Anmeldung (siehe Anhang)
- GameLoop (siehe Anhang)
- HomeBase (siehe Anhang)

Erstellen einer ausführbaren Datei

- Zum Erstellen einer ausführbaren Datei für das Spiel „Fang den Hut“ ist die installierte Entwicklungsumgebung „Microsoft Visual Studio C++“ in Version 6 erforderlich
- Erstellen sie an einer beliebigen Stelle auf ihrer Festplatte einen Projektordner z.B C:\ FangDenHut
- Kopieren Sie den Inhalt des Ordners [CDROM]:\FangDenHut\src in den von ihnen angelegten Projektordner
- In dem Projektordner befindet sich eine Datei FangDenHut.dsw. Nach einem Doppel-Klick auf diese Datei öffnet sich das Projekt „FangDenHut“ in der Entwicklungsumgebung
- Vergewissern sie sich, daß die richtige Projektkonfiguration – Release - eingestellt ist. Dazu im Menü den Punkt „Erstellen -> Aktive Konfiguration festlegen …“ anklicken. Im geöffneten Fenster „FangDenHut-Win32 Release“ auswählen und mit einem Klick auf den OK-Button die Auswahl bestätigen
- Zur Anwendungserstellung im Menü „Erstellen“ den Punkt „FangDenHut.exe“ anklicken. Daraufhin startet der Kompiliervorgang
- Wurde die Anwendung erfolgreich kompiliert und erstellt, so befindet sich in dem Projektordner ein Verzeichnis „Release“ in dem sich die ausführbare Datei „FangDenHut.exe“ befindet

Anhang

- A. UseCase – Simple (Diagramm & Beschreibung)
- B. UseCase – Complex (Diagramm & Beschreibung)
- C. Klassendiagramm & Klassenbeschreibungen
- D. Sequenzdiagramm – Anmeldung
- E. Sequenzdiagramm – GameLoop
- F. Sequenzdiagramm – HomeBase

A. UseCase – Simple (Diagramm & Beschreibung)

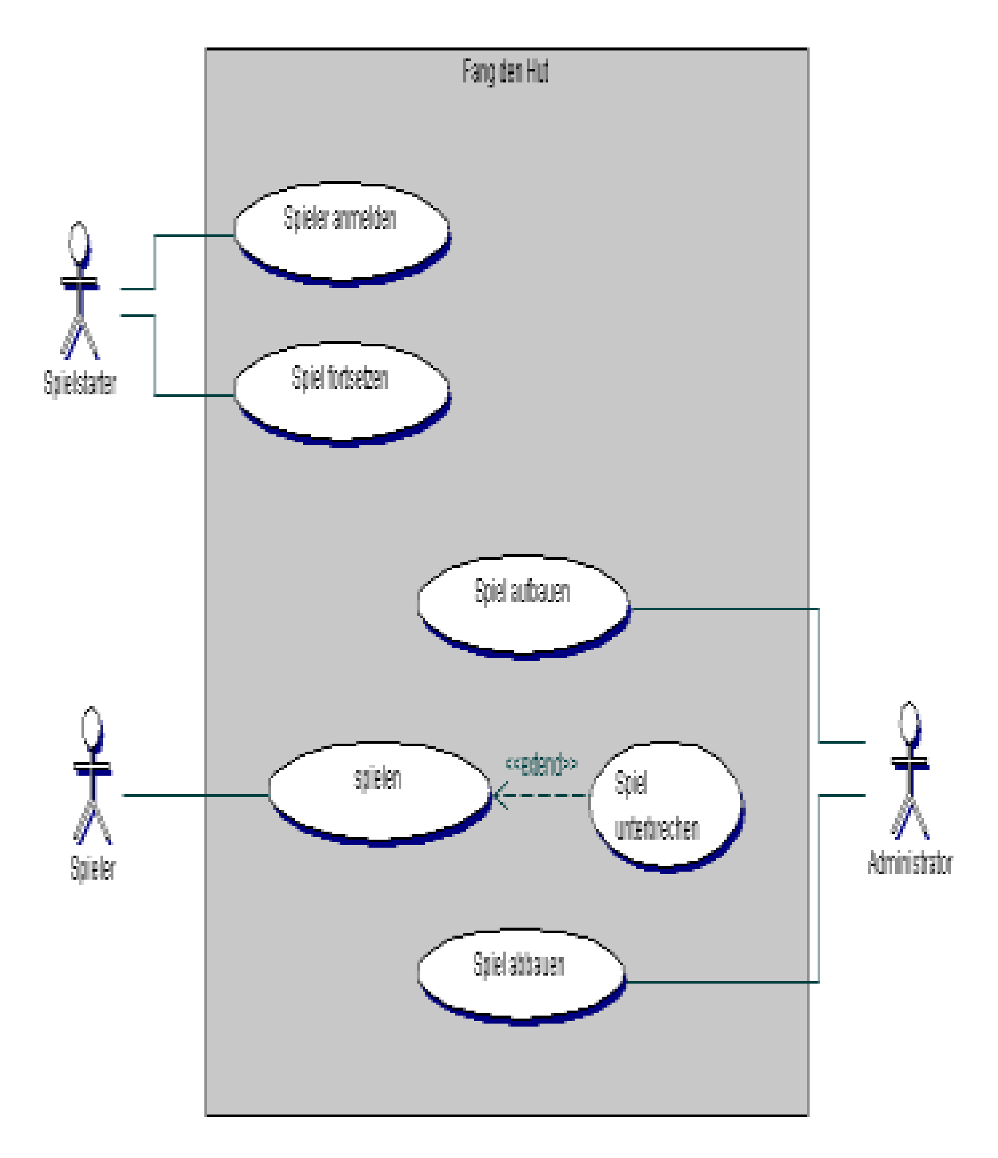
Fang den Hut
Spielstarter
Spieler anmelden
Spiel fortsetzen
Spiel aufbauen
Spieler
spielen
<<extend>>
Spiel unterbrechen
Spiel abbauen
Administrator

Package <default>

Use Case Diagram Summary	
FDH−Simple	Dies ist das gesamte Spiel "Fang den Hut" in einfachster Darstellung.

<default>

Use Case Diagram FDH−Simple



Dies ist das gesamte Spiel "Fang den Hut" in einfachster Darstellung. Detailiertere Darstellung im Diagramm "FDH−Comlex".

Actor Summary	
Administrator	Der "Administrator" ist der Computer selber.
Spieler	Dies ist der eigentliche Spieler von welchen immer 2−4 an einem Spiel teilnehmen.
Spielstarter	Dieser Aktor repräsentiert die Person die die Spielstartvorbereitungnen trifft.

System Boundary Summary	
Fang den Hut	Dies ist das gesamte Spiel "Fang den Hut" in einfachster Darstellung.

Diagram Elements Detail

Administrator

Der "Administrator" ist der Computer selber. Er kümmert sich um den Auf− und Abbau des Spielbretts und aller anderen Objekte.

Communicates Links

to UseCase Spiel aufbauen

Spieler

Dies ist der eigentliche Spieler von welchen immer 2−4 an einem Spiel teilnehmen. Diese wechseln sich nacheinander ab.

Communicates Links

to UseCase spielen

Spielstarter

Dieser Aktor repräsentiert die Person die die Spielstartvorbereitungnen trifft. Sie kann ein neues Spiel starten und muss dann alle Mitspieler anmelden, oder sie kann ein unterbrochenes Spiel wiederaufnehmen.

Communicates Links

to UseCase Spieler anmelden

to UseCase Spiel fortsetzen

Fang den Hut

Dies ist das gesamte Spiel "Fang den Hut" in einfachster Darstellung. Detailiertere Darstellung im Diagramm "FDH−Comlex".

backgroundColor 200,200,200

UseCase Summary	
Spiel abbauen	Das Spielbrett sowie alle anderen Spielobjekte werden entfernt.
Spiel aufbauen	Das Spielbrett sowie alle Spielobjekte werden initialisiert, entweder mit den normalen Startwerten (wenn ein neues Spiel gestartet wird) oder mit den Daten des unterbrochenen Spiels (so dass es komplett wiederhergestellt werden kann).
spielen	Das eigentliche Spielen von "Fang den Hut".
Spieler anmelden	Hier werden alle Mitspieler registriert.
Spiel fortsetzen	Ein vorher unterbrochenenes Spiel wird fortgesetzt.
Spiel unterbrechen	Der aktuelle Status des Spiels wird gespeichert so dass es komplett rekonstruiert werden kann.

Spiel abbauen

Das Spielbrett sowie alle anderen Spielobjekte werden entfernt.

Communicates Links

to Actor Administrator

Spiel aufbauen

Das Spielbrett sowie alle Spielobjekte werden initialisiert, entweder mit den normalen Startwerten (wenn ein neues Spiel gestartet wird) oder mit den Daten des unterbrochenen Spiels (so dass es komplett wiederhergestellt werden kann).

---

spielen

Das eigentliche Spielen von "Fang den Hut". Jeder Spieler kommt nacheinander dran. Nach einem Zug besteht die Möglichkeit das <Spiel zu unterbrechen>.

Spieler anmelden

Hier werden alle Mitspieler registriert. Nur registrierte Spieler dürfen auch spielen. Es muss ausserdem auch auf die Minimal− und Maximalspieleranzahl geachtet werden (2−4). Jeder der Mitspielenden bekommt eine eindeutig Farbe zugewiesen anhand welcher er seine Spielobjekte identifizieren kann.

Spiel fortsetzen

Ein vorher unterbrochenenes Spiel wird fortgesetzt. Das gesamte Spiel muss dafür vollständig rekonstruiert werden.

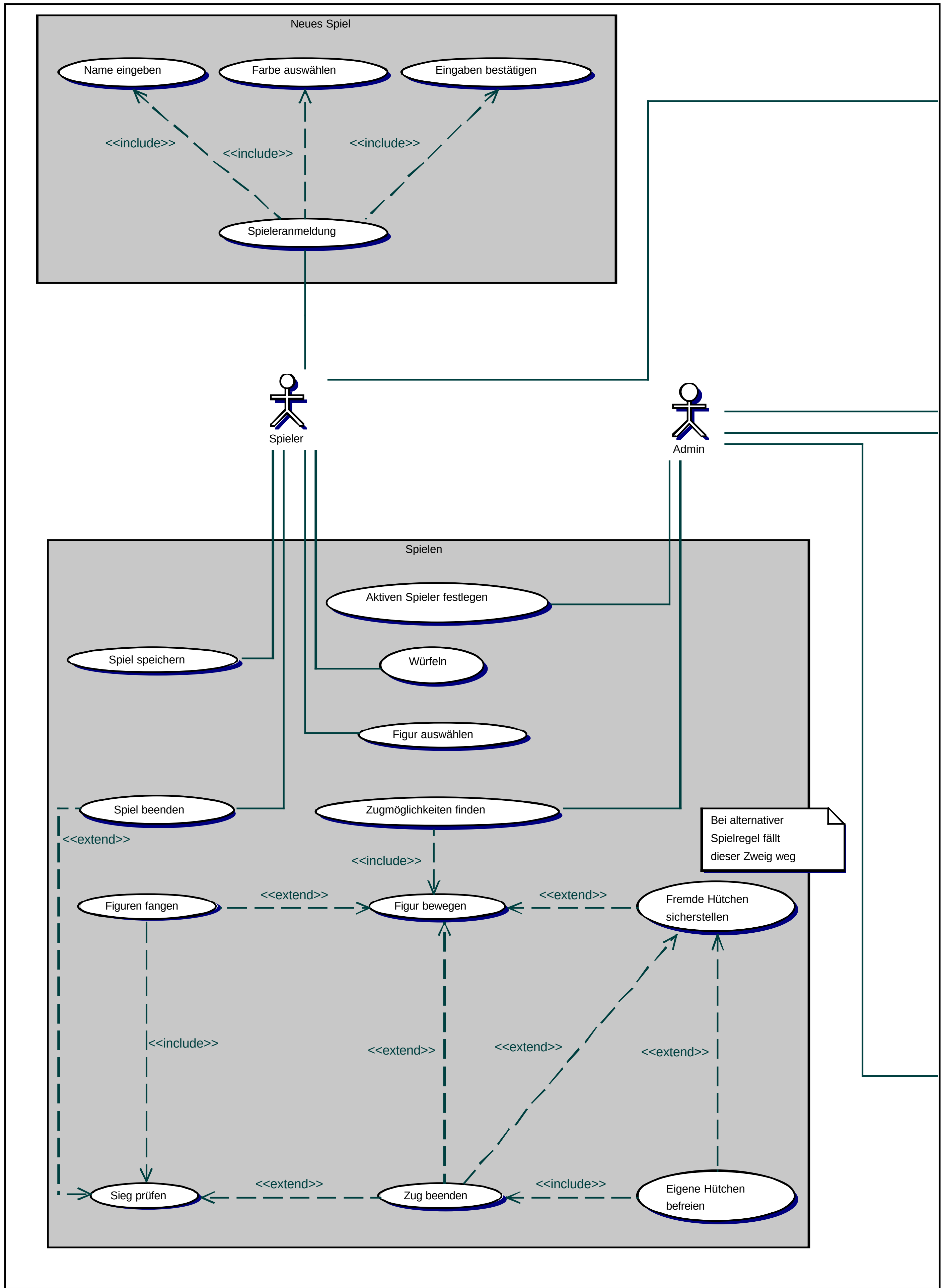
Spiel unterbrechen

Der aktuelle Status des Spiels wird gespeichert so dass es komplett rekonstruiert werden kann.

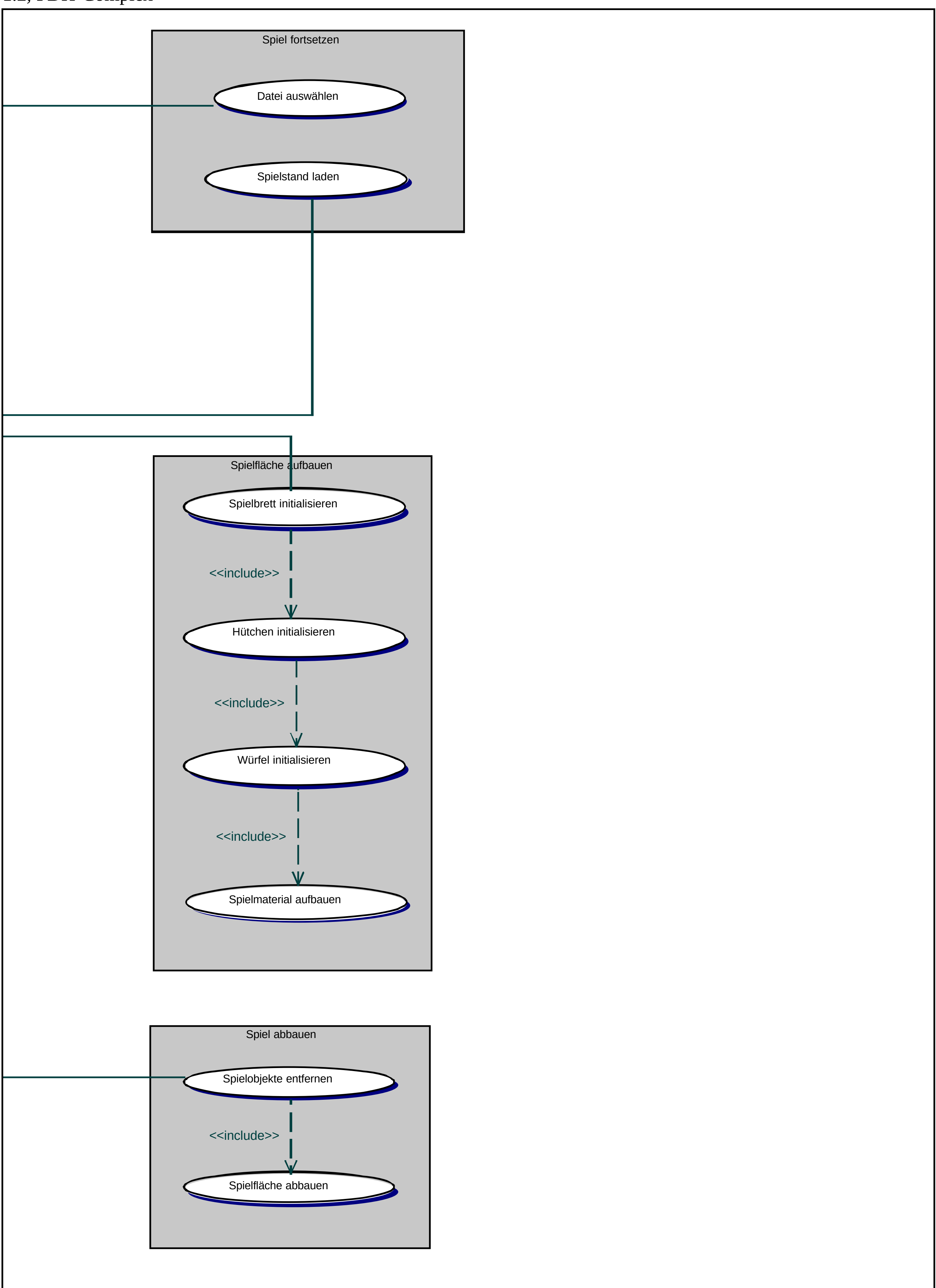
Extends Links

to UseCase spielen

B. UseCase – Complex (Diagramm & Beschreibung)



Neues Spiel
Name eingeben
Farbe auswählen
Eingaben bestätigen
<<include>>
<<include>>
<<include>>
Spieleranmeldung
Spieler
Admin
Spielen
Aktiven Spieler festlegen
Spiel speichern
Würfeln
Figur auswählen
Spiel beenden
Zugmöglichkeiten finden
Bei alternativer
Spielregel fällt
dieser Zweig weg
<<extend>>
<<include>>
Figuren fangen
<<extend>>
Figur bewegen
<<extend>>
Fremde Hütchen
sicherstellen
<<include>>
<<extend>>
<<extend>>
<<extend>>
Sieg prüfen
<<extend>>
Zug beenden
<<include>>
Eigene Hütchen
befreien

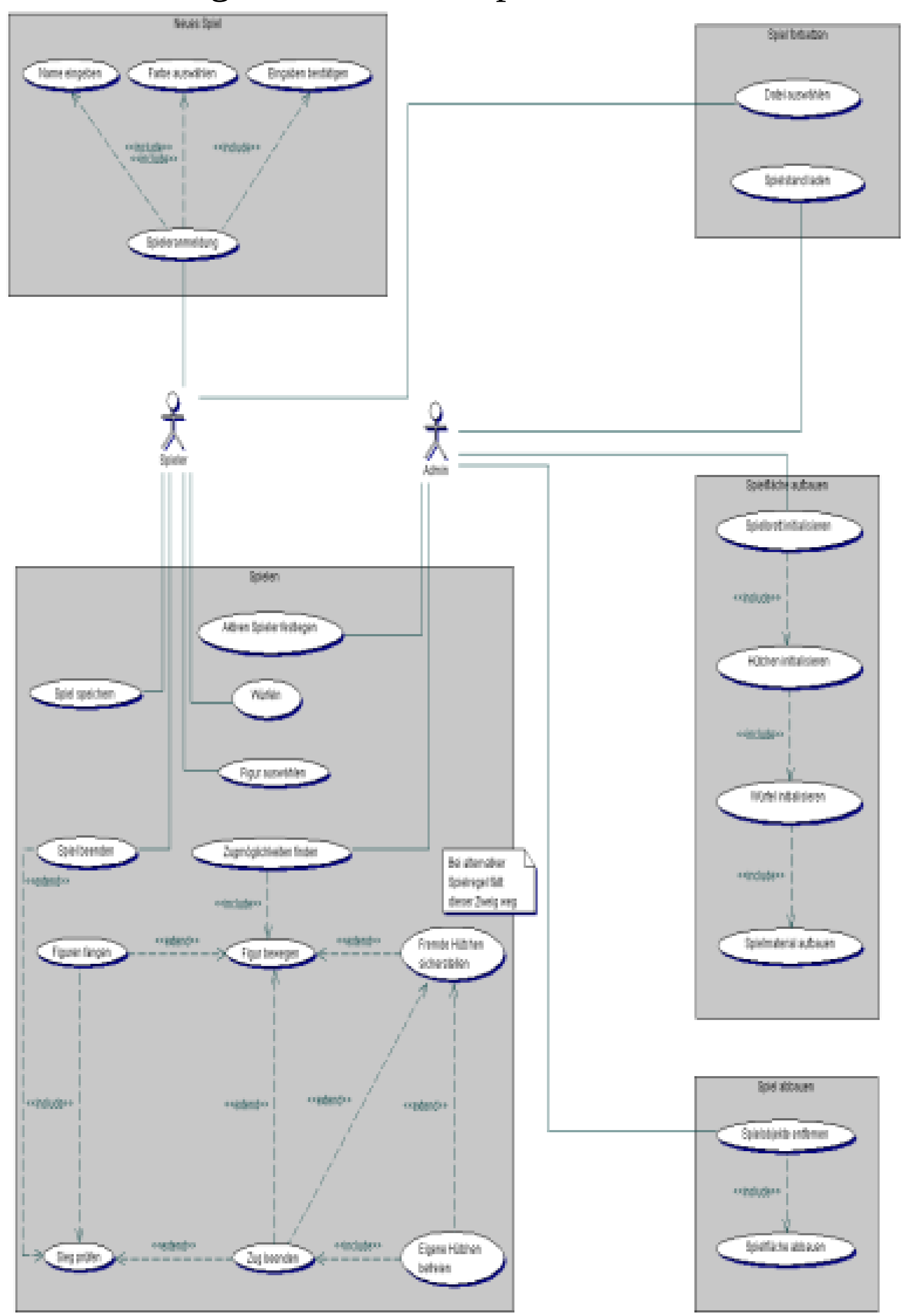

Spiel fortsetzen
Datei auswählen
Spielstand laden
Spielfläche aufbauen
Spielbrett initialisieren
<<include>>
Hütchen initialisieren
<<include>>
Würfel initialisieren
<<include>>
Spielmaterial aufbauen
Spiel abbauen
Spielobjekte entfernen
<<include>>
Spielfläche abbauen

Package <default>

Use Case Diagram Summary	
FDH−Complex	In diesem UseCase−Diagramm werden in detailierter Form die gesamten einzelnen Verhalten die im Spiel "Fang den Hut" autreten können dargestellt, sowie die Zuständigkeiten der einzelnen Akteure.

<default>

Use Case Diagram FDH−Complex



In diesem UseCase−Diagramm werden in detailierter Form die gesamten einzelnen Verhalten die im Spiel "Fang den Hut" autreten können dargestellt, sowie die Zuständigkeiten der einzelnen Akteure.

Actor Summary	
Admin	Der Administrator stellt das System dar und übernimmt die Spielführerrolle.
Spieler	Dieser Actor repräsentiert die Spieler und führt stellvertretend deren Aktionen aus

Note Summary	
Note1	

System Boundary Summary	
Neues Spiel	Dieser UseCase stellt den allgemeinen Ablauf beim Starten dar.
Spiel abbauen	Beim Beenden eines Spiels müssen Datenstrukturen gelöscht und Speicher freigegeben werden.
Spielen	Während des Spiels erfolgt die Darstellung auf einem Spielfeldt.
Spielfläche aufbauen	Zu beginn eines Spiels mus die Spielanzeige aufgebaut und dargestellt werden.
Spiel fortsetzen	Eine unterbrochene/gespeicherte Partie Fang den Hut kann zu einem späteren Zeitpunkt fortsetzt werden.

Diagram Elements Detail

Admin

Der Administrator stellt das System dar und übernimmt die Spielführerrolle.

Communicates Links

to UseCase Spielbrett initialisieren

to UseCase Aktiven Spieler festlegen

to UseCase Zugmöglichkeiten finden

to UseCase Spielobjekte entfernen

to UseCase Spielstand laden

Spieler

Dieser Actor repräsentiert die Spieler und führt stellvertretend deren Aktionen aus

Communicates Links

to UseCase Datei auswählen

to UseCase Würfeln

to UseCase Figur auswählen

to UseCase Spiel speichern

to UseCase Spiel beenden

to UseCase Spieleranmeldung

Note1

text Bei alternativer
Spielregel fällt
dieser Zweig weg

Neues Spiel

Dieser UseCase stellt den allgemeinen Ablauf beim Starten dar. Der Spieler startet das "Fang den Hut"−Programm und bekommt den Anmeldebildschirm angezeigt. Dort nimmt er die nötigen Einstellungen für ein neues Spiel vor. Es werden für 2 bis 4 Spieler die Namen eingegeben und ihre Farben ausgewählt. Dabei erhält jeder Spieler eine andere Farbe. Weiterhin hat man die Möglichkeit zwischen den normalen und den alternativen Regeln zu entscheiden. Wurden alle Eingaben richtig getätigt, so erscheint das "Fang den Hut" Spielfeld und das Spiel beginnt. Wurden bei der Eingabe Fehler gemacht, so erscheint eine Fehlermeldung. (Häufigste Fehler: nur ein Spieler ausgewählt, Farbe mehrfach vergeben)

backgroundColor 200,200,200

UseCase Summary	
Eingaben bestätigen	VIEW/CONTROLLER: Warnung falls weniger als 2 Spielernamen eingegeben, da das Spiel erst ab 2 Spieler spielbar ist.

UseCase Summary	
Farbe auswählen	VIEW/CONTROLLER: Pro Spieler Auswahl einer der 4 Farben (blau, rot, grün, gelb) über 4 Radiobuttons; temporäre Speicherung über Membervariable/View; ausgewählte Farbe wird für andere Spieler deaktiviert
Name eingeben	VIEW: Eingabe der Namen über Textfelder, temporäre Speicherung in Membervariablen des Views; 4 Namen = Spieler
Spieleranmeldung	VIEW: Darstellung der Anmeldemaske und temporäre Speicherung der Benutzereingaben CONTROLLER: Erstellen von Benutzerobjekten und Speichern dieser im Document.

Eingaben bestätigen

VIEW/CONTROLLER:
Warnung falls weniger als 2 Spielernamen eingegeben, da das Spiel erst ab 2 Spieler spielbar ist.
Aufruf des Document mit allen eingegebenen Informationen
,
Aufruf von "InitGame()"

DOCUMENT:
Erstellung von Spieler mit den übergebenen Daten;
Spieler in statisches Array (4) einfügen/verlinken;
private Variable array_pos

Farbe auswählen

VIEW/CONTROLLER:
Pro Spieler Auswahl einer der 4 Farben (blau, rot, grün, gelb) über 4 Radiobuttons; temporäre Speicherung über Membervariable/View;
ausgewählte Farbe wird für andere Spieler deaktiviert

Name eingeben

VIEW:
Eingabe der Namen über Textfelder, temporäre Speicherung in Membervariablen des Views;
4 Namen = Spieler

Spieleranmeldung

VIEW: Darstellung der Anmeldemaske und temporäre Speicherung der Benutzereingaben

CONTROLLER:
Erstellen von Benutzerobjekten und Speichern dieser im Document.

DOCUMENT:
Ablegen der Benutzerobjekte

Includes Links

to UseCase Name eingeben

to UseCase Farbe auswählen

to UseCase Eingaben bestätigen

Spiel abbauen

Beim Beenden eines Spiels müssen Datenstrukturen gelöscht und Speicher freigegeben werden.

backgroundColor 200,200,200

UseCase Summary	
Spielfläche abbauen	VIEW : Darstellung aktualisieren.
Spielobjekte entfernen	DOCUMENT : Löschen der Hütchen und des Spielfeldes.

Spielfläche abbauen

VIEW :
Darstellung aktualisieren.
 Zurück zur Starmaske.

Spielobjekte entfernen

DOCUMENT :
Löschen der Hütchen und des Spielfeldes.
Statistiken aktualisieren.(Highscores)
Löschen der Spieler und Spielerliste.

Includes Links

to UseCase Spielfläche abbauen

Spielen

Während des Spiels erfolgt die Darstellung auf einem Spielfeldt. Auf einem Spielbrett werden die Hüte angezeigt, außerhalb werden zusätzliche Informationen wie Spielername, Farbe und gewürfelte Augenzahl dargestellt. Das Spielbrett, besteht aus den vier Heimfeldern, den Spielfeldern, und den Hüten. Die Sicherheitsfelder sind abgedunkelt dargestellt, die Hüte werden durch farbige Kreise auf den Feldern repräsentiert.
Zu Beginn des Spiels stehen alle Hüte auf den Heimfeldern. Durch Klicken auf einen Hut wird dieser ausgewählt. Sollten mehrere Hüte auf einem Feld stehen, kann man durch mehrfaches klicken die zur Verfügung stehenden Hüte durchgehen. Der ausgewählte Hut wird in einem Auswahlfenster dargestellt. Für den ausgewählten Hut werden die möglichen Zielfelder automatisch berechnet und auf dem Spielbrett farblich gekennzeichnet. Die Sicherheitsfelder werden etwas dunkler gefärbt. Durch das Klicken eines der markierten Zielfelder wird der ausgewählte Hut dorthin bewegt und der nächste Spieler ist an der Reihe. Um die Hutauswahl aufzuheben klickt man auf die rechte Maustaste oder einfach auf einen der anderen Spielerhüte. Die gültigen Spielfelder für diesen Hut werden dann sofort angezeigt.

Hat ein Spieler eine 6 gewürfelt, dann führt er seinen Spielzug wie gewohnt aus und ist danach ein weiteres Mal dran.

Landet ein Hut auf einem Feld auf dem sich Hüte anderer Spieler befinden so werden die fremden Hüte von dem aktiven Hut des aktiven Spielers gefangen genommen.
Sonderfall: Auf einem Sicherheitsfeld können keine Hüte gefangen werden.

In den normalen Spielregeln besteht die Möglichkeit, das ein Hut seine Gefangenen zu Hause abliefert (sicherstellt). Diese werden dann aus dem Spiel genommen. Sollten sich unter den Gefangenen einige eigene Hüte befinden, werden diese befreit und auf dem Heimatfeld des Spielers plaziert. Ein Hut kann nur nach Hause zurück, wenn er Gefangene besitzt. Es reichts eine genügend Hohe Augenzahl gewürfelt zu haben um das Heimatfeld zu erreichen. Die einzige Besonderheit besteht nun darin, dass übrige Augen nun ebenfalls verbraucht werden müssen, d.h. der Spielzug wird fortgesetzt. Ein Hutwechsel ist in dieser Situation nicht möglich. Dieser Zugzwang besteht nicht, wenn das Heimatfeld mit einer genau passenden Augenzahl erreicht wurde.
Bei den alternativen Spielregeln entfällt dieser Spielzug.

Eine Partie Fang den Hut kann unterbrochen und zu einem späteren Zeitpunkt fortsetzt werden. Ein Spielstand kann unter einem beliebigen Namen gespeichert werden. Ein Spielstand kann zu Beginn des Spiels, oder mitten während eines laufenden Spiels geladen werden.

backgroundColor 200,200,200

UseCase Summary	
Aktiven Spieler festlegen	CONTROLLER.
Eigene Hütchen befreien	DOCUMENT: Die eigenen Hütchenobjekte werden aus der Liste des ehemaligen Fängers entfernt, und in die Liste des ursprünglichen Besitzers eingefügt.
Figur auswählen	Klick auf den gewünschten Hut hat stattgefunden.
Figur bewegen	Klick auf legales Zielfeld hat stattgefunden. (illegale Feldklicks werden vom VIEW ignoriert) DOCUMENT: Spielerstatus auf "passiv" setzen.
Figuren fangen	VIEW: Die Darstellung wird aktualisiert.
Fremde Hütchen sicherstellen	DOCUMENT: Sofern fremde Hütchen mitgeführt wurden, werden diese aus der Opferliste des aktuellen Hütchens entfernt.
Sieg prüfen	CONTROLLER: Wenn andere Spieler keine verfügbare Figuren mehr haben − mit "Spiel beenden" fortfahren, sonst mit "Zug beenden".
Spiel beenden	Wurde festgestellt das ein Spieler gewonnen hat, so kann das Spiel beendet werden.
Spiel speichern	Ein Spiel kann zwischen zwei Spielzügen gespeichert werden.

UseCase Summary	
Würfeln	Klick auf Würfelbutton hat stattgefunden.
Zug beenden	CONTROLLER: Hat der aktive Spieler in diesem Zug eine 6 gewürfelt − mit "Würfeln" fortfahren, Aufruf von DOCUMENT −> Spielerzeiger weitersetzen.
Zugmöglichkeiten finden	VIEW : Aufruf von DOCUMENT −> Zugmöglichkeiten finden.

Aktiven Spieler festlegen

CONTROLLER.
Entscheidung (1. Zug − 1. Zug nach Laden − Folgezug)

1.Zug : Aufruf an DOCUMENT −> Spielerliste mischen und Positionszeiger auf 1 setzen.

1.Zug nach Laden : Spielerliste ist bereits richtig.
Spieler am Positionszeiger ist drann.

Folgezug : Spielerliste ist bereits richtig.
Spieler am Positionszeiger ist drann.

Die Spielerposition wird bei "Spielzug beenden" aktualisiert.
So wird sichergestellt, dass beim Speichern
der Positionszeiger richtig steht.

DOCUMENT:
Führt die Aufrufe des CONTROLLERS aus.

VIEW:
Aktiven Spieler "markieren"; durch Rahmen, oder durch übertragen der Spielerdaten in ein spezieles Fenster.
Button "Würfeln" aktivieren.

Eigene Hütchen befreien

DOCUMENT:

Die eigenen Hütchenobjekte werden aus der Liste des ehemaligen Fängers entfernt, und in die Liste des ursprünglichen Besitzers eingefügt.
Der Status der befreiten Hütchen wird aktualisiert.
Die Positionszeiger der befreiten Hütchen werden auf HomeBase gesetzt, und die Liste des Feldes wird akutalisiert.
(das Feld enthält jetzt alle eigenen und alle fremden Hütchen)

VIEW:
Die Darstellung wird aktualisiert.
Die befreiten Hütchen werden auf HomeBase dargestellt.

CONTROLLER:
Mit "Zug beenden" fortfahren.

Extends Links

to UseCase Fremde Hütchen sicherstellen

Includes Links

to UseCase Zug beenden

Figur auswählen

Klick auf den gewünschten Hut hat stattgefunden.

VIEW/DOCUMENT:
Hütchenobjekt aus Mauskoordinaten bestimmen.
Gewähltes Hütchen an DOCUMENT übermitteln.
Hütchen "markieren".
Illegale Auswahlen müssen ignoriert werden.

Figur bewegen

Klick auf legales Zielfeld hat stattgefunden.
(illegale Feldklicks werden vom VIEW ignoriert)

DOCUMENT:
Spielerstatus auf "passiv" setzen.

VIEW:
Dem Document die gewählte Position übermitteln.
Darstellung des Bewegten Hütchens wird erst durchgeführt, wenn klar ist, was auf dem Feld passiert.

CONTROLLER:
Fragen an DOCUMENT zum Zielfeld.
Nächsten Schritt entsprechend den Regeln prüfen:

− Befindet sich der Hut auf einem Zielfeld/Versteck − mit "Fremde Hütchen sicherstellen" fortfahren. (B alternativer Spielregel fällt diese Möglichkeit weg. In diesem Fall mit "Zug beenden" fortfahren.)

− Befindet sich der Hut auf einem Normalfeld auf dem bereits fremde Hütchen stehen − mit "Figuren fangen" fortfahren.

Figuren fangen

VIEW:
Die Darstellung wird aktualisiert.
Es werden das/die gefangene(n) Hütchen entfernt (status "gefangen"),
und das bewegte Hütchen auf dem Feld positioniert.
Aufruf von DOCUMENT −> figuren fangen.

DOCUMENT:
Figuren fangen :
Die gefangenen Hütchenobjekte werden aus der
Liste des Spielerobjekts entfernt, und in die Opferliste des bewegten Hütchens eingesetzt.
(Fanglist aktualisieren)
Der Status der gefangenen Hütchen wird aktualisiert.
Die Liste des Spielfeldes wird aktualisiert.
Falls ein Spieler seine letzten Hütchen verliert Spielerstatus auf "ausgeschieden" setzen.

CONTROLLER:
Mit "Sieg prüfen" fortfahren.

Extends Links

to UseCase Figur bewegen

Includes Links

to UseCase Sieg prüfen

Fremde Hütchen sicherstellen

DOCUMENT:

Sofern fremde Hütchen mitgeführt wurden,
 werden diese aus der Opferliste des aktuellen Hütchens entfernt.
 Ihr Status wechselt zu "Knast", und sie werden in die Liste des Spielfeldes integriert.
 (Punkte aktualisieren)

CONTROLLER:
Hat der aktive Hut eigene Hütte in "Gefangenschaft" − mit "eigene Hütte befreien" fortfahren.

VIEW:
Die Darstellung wird aktalisiert.
 Das bewegte und alle fremden mitgeführten Hütchen, werden auf das Starfeld gesetzt. (fremde Hütchen werden von eigenen getrennt dargestellt).
 Fremde Hütchen im "Gefängnis" können nicht ausgewählt werden, werden allerdings dargestellt, im Gegensatz zu "gefangenen" Hütchen, die komplett vom Spielbrett entfernt werden.

Extends Links

to UseCase Figur bewegen

Sieg prüfen

CONTROLLER:
Wenn andere Spieler keine verfügbare Figuren mehr haben − mit "Spiel beenden" fortfahren, sonst mit "Zug beenden".

Spiel beenden

Wurde festgestellt das ein Spieler gewonnen hat, so kann das Spiel beendet werden. Ein Spiel kann zwischen zwei Zügen von einem Spieler ebenfalls beendet werden.

VIEW: Meldung "Spielende" anzeigen.

CONTROLLER:
Aufräumarbeiten starten. Siehe UseCase "Spiel Abbauen"

Extends Links

to UseCase Sieg prüfen

Spiel speichern

Ein Spiel kann zwischen zwei Spielzügen gespeichert werden.

VIEW :
Speicherdialog anzeigen.
Aufruf von DOCUMENT −> Serialize−Funktion

DOCUMENT :
Serialize :
Vererbte Serialisierung von :
Spielerliste −> Spieler −> Hütchen.
Spielfeld
"Randinformationen"

Würfeln

Klick auf Würfelbutton hat stattgefunden.

VIEW:
Aufruf von DOCUMENT −> Würfeln.
Gewürfelte Augenzahl darstellen.
Würfelbutton deaktivieren.

Zug beenden

CONTROLLER:
Hat der aktive Spieler in diesem Zug eine 6 gewürfelt − mit "Würfeln" fortfahren, Aufruf von DOCUMENT −> Spielerzeiger weitersetzen.

(Die Spielerliste wird hier aktualisiert, damit beim Speichern kein Fehler auftritt)

Extends Links

to UseCase Fremde Hütchen sicherstellen

to UseCase Figur bewegen

to UseCase Sieg prüfen

Zugmöglichkeiten finden

VIEW :
Aufruf von DOCUMENT −> Zugmöglichkeiten finden.
Rückgabewert : Liste mit möglichen Feldern.
Die möglichen Zielfelder in der Darstellung hervorheben.

DOCUMENT :
Zugmöglichkeiten finden :
Feldliste vom aktuellen feld aus durchiterieren.
Mögliche Zielfelder in eine Liste schreiben.
(Achtung HomeBase kann nur vom Besitzer betreten werden)

(Rücksprung auf Figur auswählen möglich.)

Includes Links

to UseCase Figur bewegen

Spielfläche aufbauen

Zu beginn eines Spiels mus die Spielanzeige aufgebaut und dargestellt werden. Dazu müssen das Spielbrett, die Spielfelder, der Würfel und die Datenstrukturen für Felder, Hüte, Spieler, Würfel initialisiert werden. Die Felder werden sofort beim Spielbeginn und auf dem Spielbrett angeordnet.

backgroundColor 200,200,200

UseCase Summary

Hütchen initialisieren	DOCUMENT : Erstellen der Hütchenobjekte aus den ermittelten Daten.
Spielbrett initialisieren	VIEW: DOCUMENT: Instanz der Spielbrettklasse anlegen.
Spielmaterial aufbauen	VIEW : Darstellen der Spielfläche.
Würfel initialisieren	DOCUMENT : Erstellen des Würfelobjektes.

Hütchen initialisieren

DOCUMENT :
Erstellen der Hütchenobjekte aus den ermittelten Daten.
(Anzahl = 6* Spieleranzahl)
Zuordnen der Hütchen zum jeweiligen Spielerbojekt.

Hütchenobjekt :
− Typvariable für die Farbe und evtl. den kontrollierenden Spieler
− Zeiger auf das momentan besetze Spielfeld
− Liste mit den gefangenen Hütchen
− status

Includes Links

to UseCase Würfel initialisieren

Spielbrett initialisieren

VIEW:

DOCUMENT:
Instanz der Spielbrettklasse anlegen.
Aus den Daten, die durch "Spieleranmeldung" oder "Spiel fortsetzen"
ermittelt wurden.

SpielBrettklasse :
− besteht aus Spielfeldern in fester Verkettung (FDH−Layout)
(statisches Array)
− evtl. Spielfeldbild. (alternative −> selbst Zeichnen)

Unterobjekt Spielfeld :
− Spieltypvariable (HomeBase, Normal, ProtectionField)
− enthält 4 statische Zeiger, auf die Folgefelder,
oder auf NULL
− enthält Liste mit den Hütchen die auf diesem Feld stehen

Includes Links

to UseCase Hütchen initialisieren

Spielmaterial aufbauen

VIEW :

Darstellen der Spielfläche.
 (Bild laden, oder selbst zeichnen)
Darstellen der Hütchen, an ihren Positionen.
Darstellen von Randinformationen. :
− Spielernahmen
− Würfelbox
− Frags
...

Würfel initialisieren

DOCUMENT :
Erstellen des Würfelobjektes.

Includes Links

to UseCase Spielmaterial aufbauen

Spiel fortsetzen

Eine unterbrochene/gespeicherte Partie Fang den Hut kann zu einem späteren Zeitpunkt fortsetzt werden. Ein Spielstand kann zu Beginn des Spiels, oder mitten während eines laufenden Spiels geladen werden.

backgroundColor 200,200,200

UseCase Summary	
Datei auswählen	VIEW: In einer Filebox die Auswahl der möglichen Dateien, mit deren "Spiesituation" anzeigen.
Spielstand laden	DOCUMENT: Aus den gespeicherten Daten Spielzustand rekonstruieren.

Datei auswählen

VIEW:
In einer Filebox die Auswahl der möglichen Dateien, mit deren "Spiesituation" anzeigen.

Spielstand laden

DOCUMENT:
Aus den gespeicherten Daten Spielzustand rekonstruieren.
Es entstehen die gleichen Datenstrukturen wie bei "Eingabe bestätigen".
Es sind lediglich mehr Informationen vorhanden. So werden die Hütchen mit ihren Opfern und ihrer Feldposition erstellt und das Feld bekommt beim Erstellen die Hütchenpositionen übermittelt.
Danach erfolgt analog der Aufruf "InitGame()"

C. Klassendiagramm & Klassenbeschreibungen

1.1, FangDenHut Klassendiagramm

CDocument
CSaveState

-state:struct gamestate

+CSaveState
+~CSaveState
+Serialize:void
+SynchronizeData:void
#RestoreGame:void

1

CGamefield

-hats:CHat * []
-neighbours:int []
-GOID:int

+CGamefield
+~CGamefield
+AddHat:void
+GetHats:CHat * *
+RemoveHat:bool
+GetNeighbours:int []
+SetNeighbours:void
+SetGOID:void
+GetGOID:int

81

CGameboard

-gamefields:CGamefield [81]

+CGameboard
+~CGameboard
+GetGamefield:CGamefield *

1

1

CFDH Doc

-dice:CDice
-players:CPlayer * []
-gameboard:CGameboard
-active_player:POSITION
-hats:CHat * []

+CFDH_Doc
+~CFDH_Doc
+RegisterInterface:void
+AddPlayer:void
+GetPlayer:CPlayer *
+NextPlayer:int
+GetGamefield:CGamefield *
+GetHat:CHat *
+ThrowDice:int

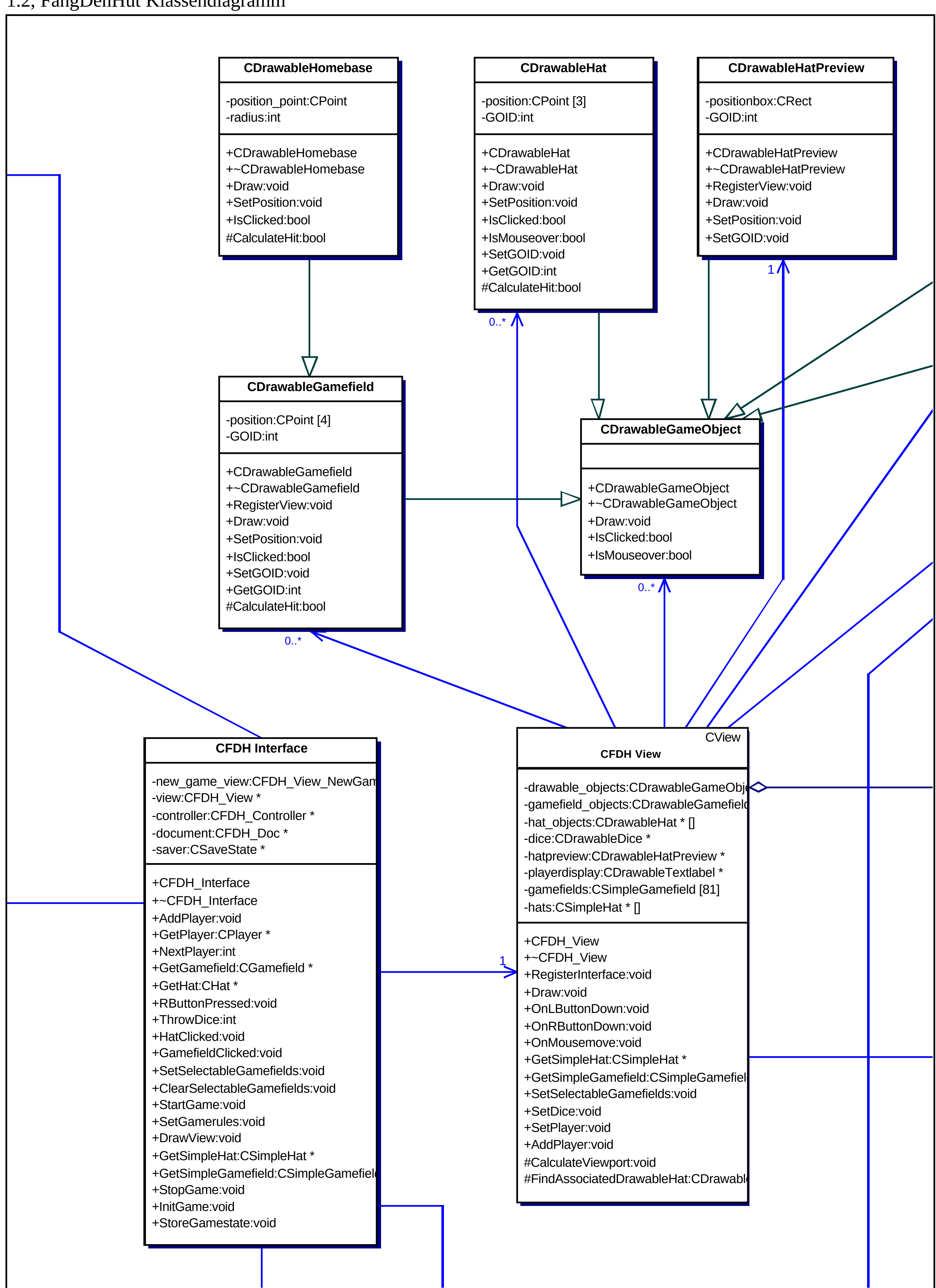
1

CDice

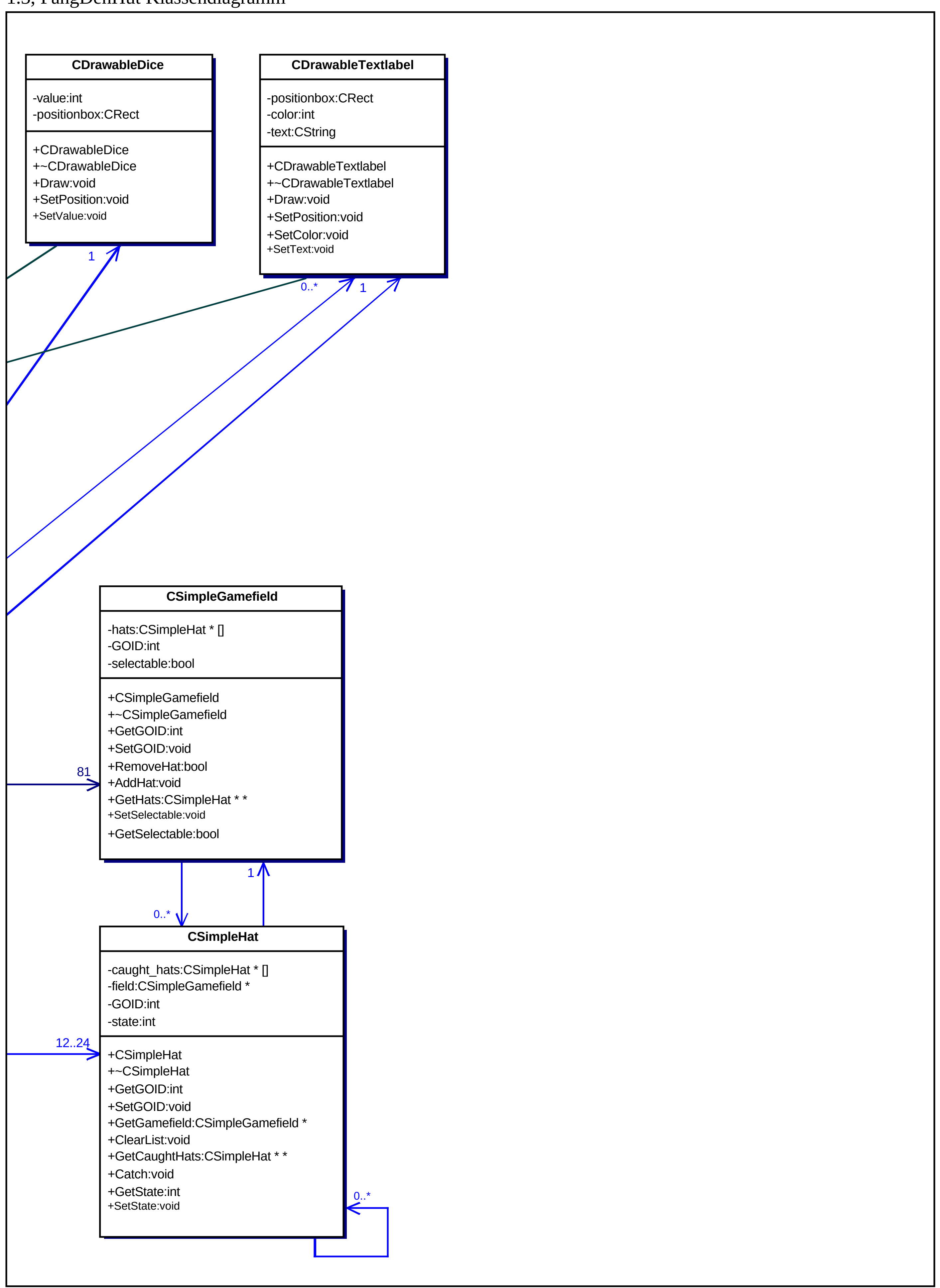
+CDice
+~CDice
+ThrowDice:int

1

1.1, FangDenHut Klassendiagramm

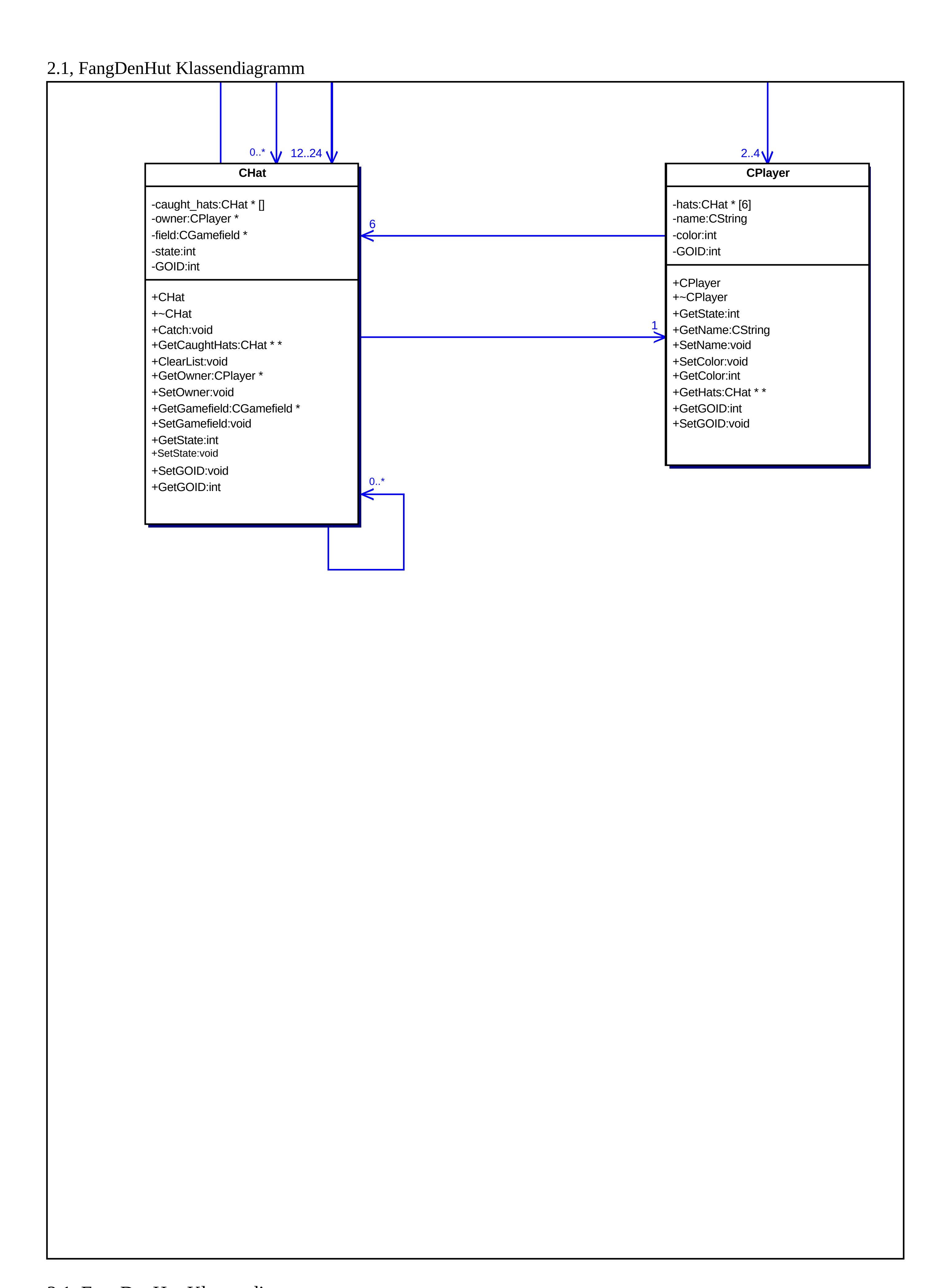

CDrawableHomebase
-position_point:CPoint
-radius:int
+CDrawableHomebase
+~CDrawableHomebase
+Draw:void
+SetPosition:void
+IsClicked:bool
#CalculateHit:bool
CDrawableHat
-position:CPoint [3]
-GOID:int
+CDrawableHat
+~CDrawableHat
+Draw:void
+SetPosition:void
+IsClicked:bool
+IsMouseover:bool
+SetGOID:void
+GetGOID:int
#CalculateHit:bool
CDrawableHatPreview
-positionbox:CRect
-GOID:int
+CDrawableHatPreview
+~CDrawableHatPreview
+RegisterView:void
+Draw:void
+SetPosition:void
+SetGOID:void
1
0..*
CDrawableGamefield
-position:CPoint [4]
-GOID:int
+CDrawableGamefield
+~CDrawableGamefield
+RegisterView:void
+Draw:void
+SetPosition:void
+IsClicked:bool
+SetGOID:void
+GetGOID:int
#CalculateHit:bool
CDrawableGameObject
+CDrawableGameObject
+~CDrawableGameObject
+Draw:void
+IsClicked:bool
+IsMouseover:bool
0..*
0..*
CFDH Interface
-new_game_view:CFDH_View_NewGam
-view:CFDH_View *
-controller:CFDH_Controller *
-document:CFDH_Doc *
-saver:CSaveState *
+CFDH_Interface
+~CFDH_Interface
+AddPlayer:void
+GetPlayer:CPlayer *
+NextPlayer:int
+GetGamefield:CGamefield *
+GetHat:CHat *
+RButtonPressed:void
+ThrowDice:int
+HatClicked:void
+GamefieldClicked:void
+SetSelectableGamefields:void
+ClearSelectableGamefields:void
+StartGame:void
+SetGamerules:void
+DrawView:void
+GetSimpleHat:CSimpleHat *
+GetSimpleGamefield:CSimpleGamefiel
+StopGame:void
+InitGame:void
+StoreGamestate:void
1
CView
CFDH View
-drawable_objects:CDrawableGameObj
-gamefield_objects:CDrawableGamefield
-hat_objects:CDrawableHat * []
-dice:CDrawableDice *
-hatpreview:CDrawableHatPreview *
-playerdisplay:CDrawableTextlabel *
-gamefields:CSimpleGamefield [81]
-hats:CSimpleHat * []
+CFDH_View
+~CFDH_View
+RegisterInterface:void
+Draw:void
+OnLButtonDown:void
+OnRButtonDown:void
+OnMousemove:void
+GetSimpleHat:CSimpleHat *
+GetSimpleGamefield:CSimpleGamefiel
+SetSelectableGamefields:void
+SetDice:void
+SetPlayer:void
+AddPlayer:void
#CalculateViewport:void
#FindAssociatedDrawableHat:CDrawabl

1.3, FangDenHut Klassendiagramm

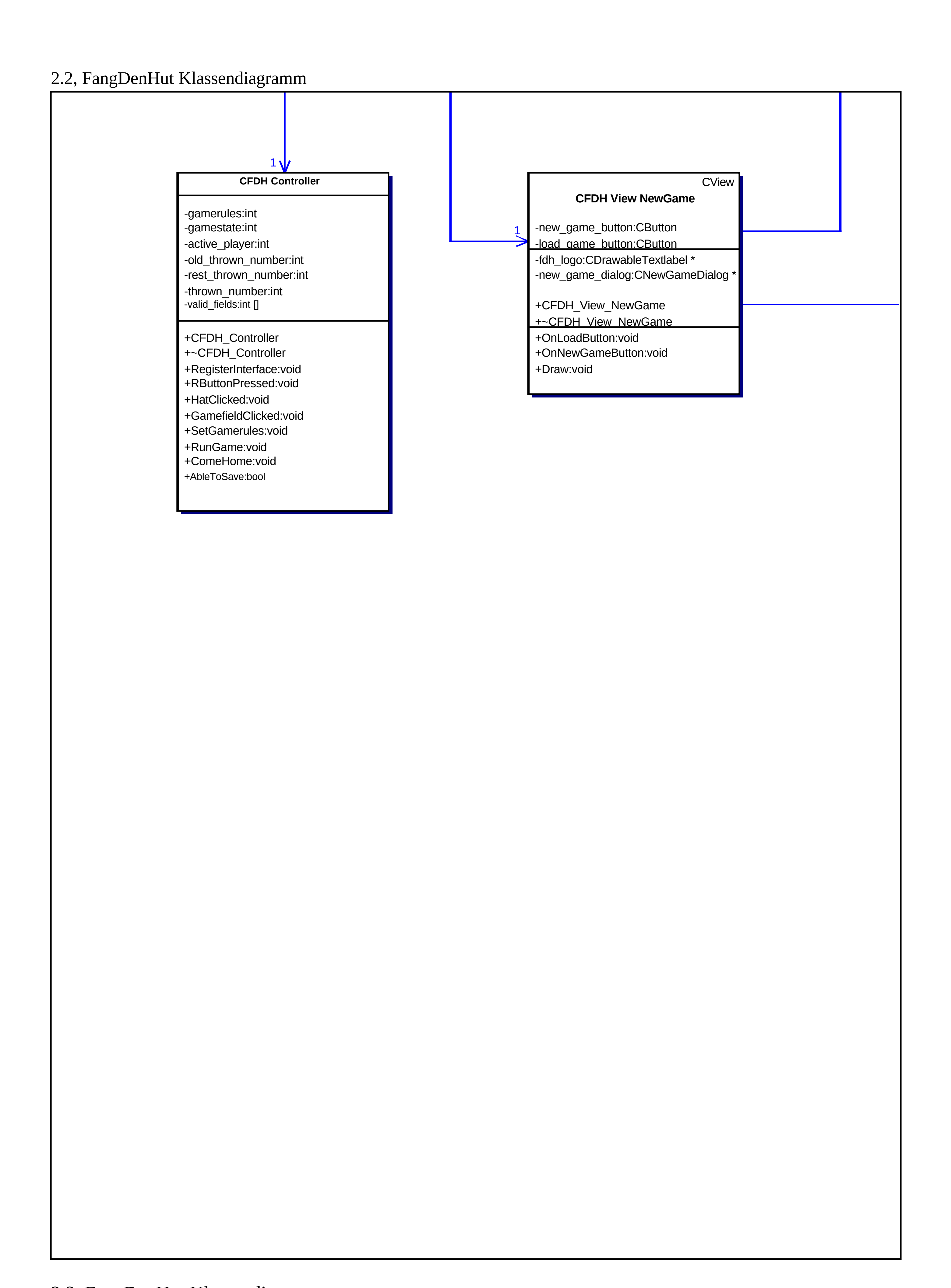


1.3, FangDenHut Klassendiagramm

2.1, FangDenHut Klassendiagramm



2.1, FangDenHut Klassendiagramm


1
CFDH Controller
-gamerules:int
-gamestate:int
-active_player:int
-old_thrown_number:int
-rest_thrown_number:int
-thrown_number:int
-valid_fields:int []
+CFDH_Controller
+~CFDH_Controller
+RegisterInterface:void
+RButtonPressed:void
+HatClicked:void
+GamefieldClicked:void
+SetGamerules:void
+RunGame:void
+ComeHome:void
+AbleToSave:bool
1
CView
CFDH View NewGame
-new_game_button:CButton
-load_game_button:CButton
-fdh_logo:CDrawableTextlabel *
-new_game_dialog:CNewGameDialog *
+CFDH_View_NewGame
+~CFDH_View_NewGame
+OnLoadButton:void
+OnNewGameButton:void
+Draw:void

1
CFormDialog
CNewGameDialog
-names:CString [4]
-colors:int [4]
-gamerules:int
-BtnCancel:CButton
-BtnOK:CButton
+CNewGameDialog
+~CNewGameDialog
+RegisterInterface:void
+OnbtnCancel:void
+OnbtnOK:void

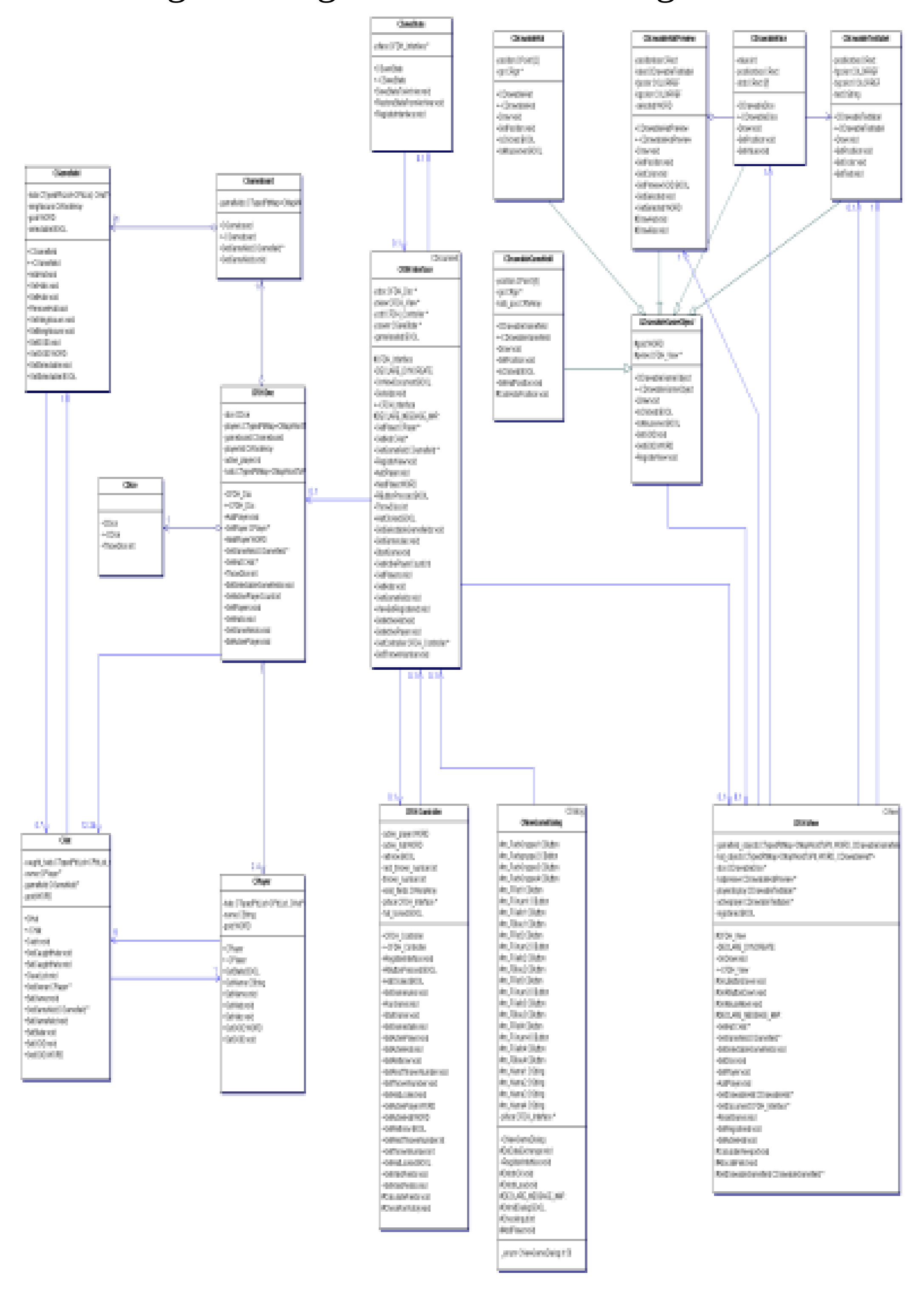
Package <default>

Class Diagram Summary	
FangDenHut Klassendiagramm	Das komplette Klassendiagramm des Spieles "Fang den Hut".

Class Summary	
CDice	Das Objekt "Würfel".
CDrawableDice	Grafikdarstellung des W?
CDrawableGamefield	Die Grafikdarstellung des Spielfeld Es kann sowohl normale Felder als auch Homebases darstellen.
CDrawableGameObject	Basisklasse für alle darstellbaren Objekte.
CDrawableHat	Grafikdarstellung eines reallen Hutobjektes.
CDrawableHatPreview	Preview eines Hutobjektes mitsamt seiner Gefangenen.
CDrawableTextlabel	Einfaches Textlabel das dargestellt werden soll
CFDH_Controller	Im CFDH−Controller sind die Spielregeln abgebildet.
CFDH_Doc	Dies ist der Document Container von FDH.
CFDH_Interface	Dies ist das zentrale Interface des Spiels.
CFDH_View	View enthält Funktionalität zur Darstellung des Spielbrettes, der Spielfiguren, des Würfels und der Spielerinformationen enthalten.
CGameboard	Das Spielbrett ist eigentlich nichts anderes als ein Container für die Spielfelder.
CGamefield	Datenobjekt Spielfeld, mit Verknüpfung auf Spielfeldnachbarn und Verknüpfung mit allen Hüten die auf dem Feld stehen.
CHat	Repräsentiert ein Spielobjekt "Hut" mit seinem Status, Besitzer, besetzes Feld und die Hüte die dieser Hut gefangen hat.
CNewGameDialog	Maske zum Starten eines neuen Spiels.
CPlayer	Repräsentiert einen Spieler in Fang den Hut.
CSaveState	CSaveState ist für die Speicherung und Wiederherstellung der Spielzustände zuständig

<default>

Class Diagram FangDenHut Klassendiagramm



Das komplette Klassendiagramm des Spieles "Fang den Hut".

Class Summary	
CDice	Imported. Das Objekt "Würfel".
CDrawableDice	Imported. Grafikdarstellung des W?

Class Summary	
CDrawableGamefield	Imported. Die Grafikdarstellung des Spielfeld Es kann sowohl normale Felder als auch Homebases darstellen.
CDrawableGameObject	Imported. Basisklasse für alle darstellbaren Objekte.
CDrawableHat	Imported. Grafikdarstellung eines reallen Hutobjektes.
CDrawableHatPreview	Imported. Preview eines Hutobjektes mitsamt seiner Gefangenen.
CDrawableTextlabel	Imported. Einfaches Textlabel das dargestellt werden soll
CFDH_Controller	Imported. Im CFDH−Controller sind die Spielregeln abgebildet.
CFDH_Doc	Imported. Dies ist der Document Container von FDH.
CFDH_Interface	Imported. Dies ist das zentrale Interface des Spiels.
CFDH_View	Imported. View enthält Funktionalität zur Darstellung des Spielbrettes, der Spielfiguren, des Würfels und der Spielerinformationen enthalten.
CGameboard	Imported. Das Spielbrett ist eigentlich nichts anderes als ein Container für die Spielfelder.
CGamefield	Imported. Datenobjekt Spielfeld, mit Verknüpfung auf Spielfeldnachbarn und Verknüpfung mit allen Hüten die auf dem Feld stehen.
CHat	Imported. Repräsentiert ein Spielobjekt "Hut" mit seinem Status, Besitzer, besetzes Feld und die Hüte die dieser Hut gefangen hat.
CNewGameDialog	Imported. Maske zum Starten eines neuen Spiels.
CPlayer	Imported. Repräsentiert einen Spieler in Fang den Hut.
CSaveState	Imported. CSaveState ist für die Speicherung und Wiederherstellung der Spielzustände zuständig

Class CDice

class CDice

Das Objekt "Würfel". Es kapselt einen Zufallsgenerator und liefert nur gültige Werte (1..6) zurück.

Constructor Summary
CDice () Initialisiert den Random−Seeds für's W?

Destructor Summary
~CDice ()

Method Summary	
`int`	**ThrowDice** () Würfelt die nächste Zahl aus und liefert diese zurück.

Constructor Detail

CDice

public **CDice** ()

Initialisiert den Random−Seeds für's W?rfeln.

Method Detail

~CDice

public synchronized **~CDice** ()

Method Detail

ThrowDice

4, CDice

public int **ThrowDice**()

Würfelt die nächste Zahl aus und liefert diese zurück.

Class CDrawableDice

---

class CDrawableDicederived from CDrawableGameObject

Grafikdarstellung des W?rfels

Darstellung
.-----. mit entsprechenden schwarzen Punkten
| o o | zur Anzeige der Würfelzahl
| o o |
'-----'

Constructor Summary
CDrawableDice ()   Standardkonstruktor

Destructor Summary	
`virtual synchronized`	**~CDrawableDice** ()

Method Summary	
`virtual void`	**Draw**(CDC * pDC) Malt den Würfel in die angegebene CRect Box.
`virtual void`	**SetPosition** (CRect & rect) setzt Position und Größe des Würfels auf dem View.
`void`	**SetValue** (int new_value) Die Zahl wird dargestellt.

Methods inherited from class CDrawableGameObject
`GetGOID, IsClicked, IsMouseover, RegisterView, SetGOID`

Constructor Detail

CDrawableDice

public **CDrawableDice** ()

Standardkonstruktor

Method Detail

~CDrawableDice

public virtual synchronized **~CDrawableDice** ()

Method Detail

Draw

public virtual void **Draw**(CDC * pDC)

Malt den Würfel in die angegebene CRect Box.

SetPosition

public virtual void **SetPosition** (CRect & rect)

setzt Position und Größe des Würfels auf dem View.

SetValue

public void **SetValue** (int new_value)

Die Zahl wird dargestellt.

Class CDrawableGamefield

class CDrawableGamefield derived from CDrawableGameObject

Die Grafikdarstellung des Spielfeld
Es kann sowohl normale Felder als auch Homebases darstellen.

Constructor Summary
CDrawableGamefield () Standardkonstruktor

Destructor Summary
~CDrawableGamefield ()

Method Summary	
`void`	**Draw**(CDC * pDC) Malt das Feld.
`BOOL`	**IsClicked** (CPoint & point) Ermittelt ob innerhalb des Objekts geklickt wurde.
`void`	**SetHatPosition** (int count, int hatsize, double * hats_x, double * hats_y, int base) Hier werden die ermittelten Hutpositionen übergeben, das Objekt erstellt daraus selbstständig CPoints
`void`	**SetPosition** (CPoint & point_ul, CPoint & point_ur, CPoint & point_br, CPoint & point_bl) Übergibt die vier Randpunkte des Spielfeldes.

Methods inherited from class CDrawableGameObject
`GetGOID, IsMouseover, RegisterView, SetGOID`

Constructor Detail

CDrawableGamefield

public **CDrawableGamefield** ()

Standardkonstruktor

Method Detail

~CDrawableGamefield

public synchronized **~CDrawableGamefield** ()

Method Detail

Draw

public void **Draw**(CDC * pDC)

Malt das Feld.

Für jedes auf ihm befindliche (simple) Hutobjekt wird view−>FindAssociateDrawableHat() aufgerufen und dann deren Position berechnet.

IsClicked

public BOOL **IsClicked** (CPoint & point)

Ermittelt ob innerhalb des Objekts geklickt wurde.

SetHatPosition

public void **SetHatPosition** (int count, int hatsize, double * hats_x, double * hats_y, int base)

Hier werden die ermittelten Hutpositionen übergeben, das Objekt erstellt daraus selbstständig CPoints

---

SetPosition

public void **SetPosition** (CPoint & point_ul, CPoint & point_ur, CPoint & point_br, CPoint & point_bl)

Übergibt die vier Randpunkte des Spielfeldes.

Class CDrawableGameObject

Direct Known Subclasses:CDrawableDice, CDrawableGamefield, CDrawableHat, CDrawableHatPreview, CDrawableTextlabel

abstract class CDrawableGameObject

Basisklasse für alle darstellbaren Objekte. Es dient für eine einheitliche Schnittstelle und stellt sicher das einige Funktionen überschrieben werden müssen. Wenn Nachfahren IsClicked() und/oder IsMouseover() nicht über− schreiben reagiert das Objekt nicht darauf.

Constructor Summary
CDrawableGameObject() Ist in der Basisklasse nur Platzhalter.

Destructor Summary	
`virtual synchronized`	**~CDrawableGameObject**() Ist in der Basisklasse nur Platzhalter.

Method Summary	
`abstract void`	**Draw**(CDC * pDC) Diese Funktion ist echt virtuell (virtual void Draw() = 0;) und MUSS in abgeleiteten Klassen überschrieben werden.
`virtual WORD`	**GetGOID**() Liefert die GOID zurück.
`virtual BOOL`	**IsClicked**(CPoint & point) Diese Funktion KANN in abgeleiteten Klassen überschrieben werden.
`virtual BOOL`	**IsMouseover**(CPoint & point) Diese Funktion KANN in abgeleiteten Klassen überschrieben werden.
`virtual void`	**RegisterView**(CFDH_View * pnew_view) Verknüpft einen neuen View
`virtual void`	**SetGOID**(WORD new_goid) Setzt die GOID.

Constructor Detail

CDrawableGameObject

public **CDrawableGameObject**()

> Ist in der Basisklasse nur Platzhalter.

Method Detail

~CDrawableGameObject

public virtual synchronized **~CDrawableGameObject**()

> Ist in der Basisklasse nur Platzhalter.

Method Detail

Draw

public abstract void **Draw**(CDC * pDC)

> Diese Funktion ist echt virtuell (virtual void Draw() = 0;) und MUSS in abgeleiteten Klassen überschrieben werden.

GetGOID

public virtual WORD **GetGOID**()

> Liefert die GOID zurück.

IsClicked

public virtual BOOL **IsClicked**(CPoint & point)

> Diese Funktion KANN in abgeleiteten Klassen überschrieben werden. Ist dies nicht der Fall, wird standardmässig false zurückgegeben (Objekt ist nicht klickbar). Ihr wird typischerweise ein CPoint−Objekt übergeben. Es muss dann vom Objekt selber überprüft werden, ob dieser Punkt in seinem klickbaren Bereich liegt oder nicht.

IsMouseover

public virtual BOOL **IsMouseover**(CPoint & point)

> Diese Funktion KANN in abgeleiteten Klassen überschrieben werden. Ist dies nicht der Fall, wird

standardmässig false zurückgegeben (Objekt reagiert auf mouseover nicht). Ihr wird typischerweise ein CPoint−Objekt übergeben. Es muss dann vom Objekt selber überprüft werden, ob dieser Punkt in seinem Reaktionsbereich liegt oder nicht.

RegisterView

public virtual void **RegisterView** (CFDH_View * pnew_view)

Verknüpft einen neuen View

SetGOID

public virtual void **SetGOID** (WORD new_goid)

Setzt die GOID.

Association Links

to Class CFDH_View

Verknüpfung zum View für Objekte die auf Funktionen zugreifen müssen.

Class CDrawableHat

---

class CDrawableHatderived from CDrawableGameObject

Grafikdarstellung eines reallen Hutobjektes. Ermöglicht Klicken und Mouseover (Preview).

Constructor Summary
CDrawableHat() Standardkonstruktor

Destructor Summary
~CDrawableHat()

Method Summary	
`void`	**Draw**(CDC * pDC) Zeichnet den Hut an die entsprechende Stelle.
`BOOL`	**IsClicked**(CPoint & point) Berechnet ob der Punkt innerhalb des Objekts liegt.
`BOOL`	**IsMouseover**(CPoint & point) Berechnet ob der Punkt innerhalb des Objekts liegt.
`void`	**SetPosition** (CPoint & point_ul, CPoint & point_br) Diese Funktion bestimmt die "Bounding Box".

Methods inherited from class CDrawableGameObject
`GetGOID, RegisterView, SetGOID`

Constructor Detail

CDrawableHat

public **CDrawableHat**()

Standardkonstruktor

Method Detail

~CDrawableHat

public synchronized **~CDrawableHat**()

Method Detail

Draw

public void **Draw**(CDC * pDC)

Zeichnet den Hut an die entsprechende Stelle. Ist der Status des Hutes gefangen oder besiegt, wird er nicht gezeichnet.

IsClicked

public BOOL **IsClicked**(CPoint & point)

Berechnet ob der Punkt innerhalb des Objekts liegt.

IsMouseover

public BOOL **IsMouseover**(CPoint & point)

Berechnet ob der Punkt innerhalb des Objekts liegt.

SetPosition

public void **SetPosition** (CPoint & point_ul, CPoint & point_br)

Diese Funktion bestimmt die "Bounding Box".

Class CDrawableHatPreview

class CDrawableHatPreview derived from CDrawableGameObject

Preview eines Hutobjektes mitsamt seiner Gefangenen.
Idee: Darstellung rechts mitte, Art der Darstellung

```
.
/ \
/_ _\
/_ _\
/_ _\  etc..
```

Constructor Summary
CDrawableHatPreview() Standardkonstruktor

Destructor Summary
~CDrawableHatPreview()

Method Summary	
void	**Draw**(CDC * pDC) Malt den Preview.
WORD	**GetSelected** () Liefert die GOID der aktuellen Selektion zurück
void	**SetColor** (COLORREF fg, COLORREF bg) Setzt die Vorder− und Hintergrundfarbe die bentutzt wird beim zeichnen
void	**SetPosition** (CRect & rect) Setzt Position und Größe.
BOOL	**SetPreviewGOID** (WORD new_goid) Setzt die darzustellende ID auf die angegebene wenn sie neu ist und gibt TRUE zurück, sie wird nicht gesetzt (und FALSE zurückgeben) wenn die angegebene ID dieselbe ist.
void	**SetSelected** (WORD new_goid) Setzt die selektierte GOID auf diesen Wert

Methods inherited from class CDrawableGameObject
`GetGOID, IsClicked, IsMouseover, RegisterView, SetGOID`

Constructor Detail

CDrawableHatPreview

public **CDrawableHatPreview**()

Standardkonstruktor

Method Detail

~CDrawableHatPreview

public synchronized **~CDrawableHatPreview**()

Method Detail

Draw

public void **Draw**(CDC * pDC)

Malt den Preview.

GetSelected

public WORD **GetSelected** ()

Liefert die GOID der aktuellen Selektion zurück

SetColor

public void **SetColor** (COLORREF fg, COLORREF bg)

Setzt die Vorder− und Hintergrundfarbe die bentutzt wird beim zeichnen

---

SetPosition

public void **SetPosition** (CRect & rect)

Setzt Position und Größe.

SetPreviewGOID

public BOOL **SetPreviewGOID** (WORD new_goid)

Setzt die darzustellende ID auf die angegebene wenn sie neu ist und gibt TRUE zurück, sie wird nicht gesetzt (und FALSE zurückgeben) wenn die
angegebene ID dieselbe ist.

SetSelected

public void **SetSelected** (WORD new_goid)

Setzt die selektierte GOID auf diesen Wert

Association Links

to Class CDrawableTextlabel

Textlabel für die Darstellung

Class CDrawableTextlabel

class CDrawableTextlabelderived from CDrawableGameObject

Einfaches Textlabel das dargestellt werden soll

Constructor Summary
CDrawableTextlabel () Standardkonstruktor

Destructor Summary	
`virtual synchronized`	**~CDrawableTextlabel** ()

Method Summary	
`virtual void`	**Draw**(CDC * pDC) Schreibt den Text.
`void`	**SetColor** (COLORREF fg, COLORREF bg) Setzt die Vorder− und Hintergrundfarbe des darzustellenden Textes.
`virtual void`	**SetPosition** (CRect & rect) Setzt die "bounding box" der Darstellung.
`void`	**SetText** (CString & new_text, enum textformat new_format) Setzt den Text der dargestellt werden soll.

Methods inherited from class CDrawableGameObject
`GetGOID, IsClicked, IsMouseover, RegisterView, SetGOID`

Constructor Detail

CDrawableTextlabel

public **CDrawableTextlabel** ()

Standardkonstruktor

Method Detail

~CDrawableTextlabel

public virtual synchronized **~CDrawableTextlabel** ()

Method Detail

Draw

public virtual void **Draw**(CDC * pDC)

Schreibt den Text.

SetColor

public void **SetColor** (COLORREF fg, COLORREF bg)

Setzt die Vorder− und Hintergrundfarbe des darzustellenden Textes.

SetPosition

public virtual void **SetPosition** (CRect & rect)

Setzt die "bounding box" der Darstellung.

SetText

public void **SetText** (CString & new_text, enum textformat new_format)

Setzt den Text der dargestellt werden soll.

Class CFDH_Controller

class CFDH_Controller

Im CFDH−Controller sind die Spielregeln abgebildet. Er entscheidet während des Spielverlaufs abhängig vo Spielzustand und den auftretenden Ereignissen welche Aktion als nächste ausgeführt werden soll bzw. darf.

Constructor Summary
CFDH_Controller ()

Destructor Summary	
`virtual synchronized`	**~CFDH_Controller** ()

Method Summary	
`const WORD`	**GetActiveHat**()
`const WORD`	**GetActivePlayer**()
`const BOOL`	**GetHatLocked**()
`const int`	**GetRestThrownNumber**()
`const BOOL`	**GetRethrow**()
`const int`	**GetThrownNumber**()
`const void`	**GetValidFields** (CWordArray & into)
`BOOL`	**HatClicked** (WORD goid) Wird aufgerufen falls ein Hut angeklickt wurde.

Method Summary	
`BOOL`	**RButtonPressed**() Wenn nach dem Auswählen einer Figur und der Anzeige der möglichen Zielfelde dieser Figur ein Rechtsklick gemacht wird, geht man einen Schritt "rückwärts" innerhalb des Spielablaufs, sprich man kann sich eine andere Figur auswählen und deren Zielfelder anzeigen lassen.
`void`	**RegisterInterface**(CFDH_Interface * pnewInterface) Registriert das ein neues Interface.
`void`	**RunGame**(WORD goid) Diese Funktion ist für den eigentlichen Spielablauf zuständig.
`void`	**SetActiveHat**(WORD hat) Setzt den aktuellen Hut
`void`	**SetActivePlayer** (WORD player) Setzt den aktuellen Spieler
`void`	**SetGamerules** (enum gamerules rules) Setzt die Spielregeln.
`void`	**SetGamestate** (enum gamestates state) Setzt den Spielstatus.
`void`	**SetHatLocked**(BOOL value)
`void`	**SetRestThrownNumber** (int value) Setzt die Anzahl der restlichen Würfelaugen.
`void`	**SetRethrow**(BOOL value) Legt fest ob ein Zug fortgesetzt wird
`void`	**SetThrownNumber** (int value) Pflegt die gewürfelte Augenzahl
`void`	**SetValidFields** (const CWordArray & src)
`void`	**StartGame**() Initialisiert alle Objekte, startet das Spiel

Constructor Detail

CFDH_Controller

public **CFDH_Controller** ()

Method Detail

~CFDH_Controller

public virtual synchronized **~CFDH_Controller** ()

Method Detail

GetActiveHat

public const WORD **GetActiveHat**()

GetActivePlayer

public const WORD **GetActivePlayer**()

GetHatLocked

public const BOOL **GetHatLocked**()

GetRestThrownNumber

public const int **GetRestThrownNumber**()

GetRethrow

public const BOOL **GetRethrow**()

GetThrownNumber

public const int **GetThrownNumber**()

GetValidFields

public const void **GetValidFields** (CWordArray & into)

HatClicked

public BOOL **HatClicked** (WORD goid)

> Wird aufgerufen falls ein Hut angeklickt wurde.
>
> Wenn gamestate = select_player:
> Es wird verglichen ob dieser Hut dem Spieler gehört, wenn nich, Abbruch, wenn doch gamestate = calculate_fields und Aufruf von RunGame()

RButtonPressed

public BOOL **RButtonPressed** ()

> Wenn nach dem Auswählen einer Figur und der Anzeige der möglichen Zielfelder dieser Figur ein Rechtsklick gemacht wird, geht man einen Schritt "rückwärts" innerhalb des Spielablaufs, sprich man kann sich eine andere Figur auswählen und deren Zielfelder anzeigen lassen.
>
> Falls gamestate == select_fields:
> gamestate = select_hat;
> ClearSelectableFields()
> empty(valid_fields)

RegisterInterface

public void **RegisterInterface** (CFDH_Interface * pnewInterface)

> Registriert das ein neues Interface. (Membervariable zur Speicherun nicht integriert, da TCC Bug (siehe Klassendescription))

RunGame

public void **RunGame**(WORD goid)

> Diese Funktion ist für den eigentlichen Spielablauf zuständig.
>
> Spielablauf:

1. gamestate = select_player;
2. Über NextPlayer() schauen ob es wieder derselbe ist −> Spiel beendet; ansonsten active_player aktualisieren
3. gamestate = dicing;
4. ThrowDice();
5. gamestate = select_hat;
(hier hört die Funktion auf, und es wird gewartet, dass HatClicked aufgerufen wird; ausserdem wird ein DrawView() gemacht)

6. gamestate == calculate_fields
Das Feld auf dem der angeklickte Hut steht wird geholt (GetHat(GOID)−>GetGamefield()) und dessen Nachbarfelder berechnet anhand der Nachbarn und der Augenzahl (restliche Augenzahl speichern bei Homebase). Ausserdem nur eigene Homebase zulassen (anhand GOID) und nur wenn Hut > 0 Gefangene (GetCaughtHats()).
gamestate = select_field
Aufruf SetSelectableField() und DrawView()
(Beenden der Funktion −> Warten auf einen Aufruf von GamefieldClicked())

7. gamestate = process_field:
Schauen ob Feld = gültige Homebase, wenn ja:
Eigene gefangene Hüte befreien, fremde Hüte komplett aus dem Spiel nehmen.
erneut bei gamestate = calculate_fields mit thrown_number = rest_number weitermachen.
Ansonsten prüfen ob Feld Ruhefeld, wenn nicht dann feindliche Hüte gefangennehmen.
8. Falls thrown_dice == 6, gamestate = dicing und dort weitermachen, ansonsten gamestate = select_player (dort wird auch auf Sieg geprüft).

SetActiveHat

public void **SetActiveHat**(WORD hat)

Setzt den aktuellen Hut

SetActivePlayer

public void **SetActivePlayer** (WORD player)

Setzt den aktuellen Spieler

SetGamerules

public void **SetGamerules** (enum gamerules rules)

Setzt die Spielregeln.

SetGamestate

public void **SetGamestate** (enum gamestates state)

Setzt den Spielstatus.

SetHatLocked

public void **SetHatLocked**(BOOL value)

SetRestThrownNumber

public void **SetRestThrownNumber** (int value)

Setzt die Anzahl der restlichen Würfelaugen.

SetRethrow

public void **SetRethrow**(BOOL value)

Legt fest ob ein Zug fortgesetzt wird

SetThrownNumber

public void **SetThrownNumber** (int value)

Pflegt die gewürfelte Augenzahl

SetValidFields

public void **SetValidFields** (const CWordArray & src)

StartGame

public void **StartGame**()

Initialisiert alle Objekte, startet das Spiel

Association Links

25, CFDH_Controller

to Class CFDH_Interface

Verknüpfung zum Interface

Class CFDH_Doc

class CFDH_Doc

Dies ist der Document Container von FDH. Es enthält alle (Daten)−Objekte, organisiert und verwaltet sie.

Constructor Summary
CFDH_Doc() Erstellt ein Spielfeld und einen Würfel.

Destructor Summary
~CFDH_Doc()

Method Summary	
`void`	**AddPlayer**(CString & name, enum playercolor color) Registriert einen neuen Spieler.
`int`	**GetActivePlayerCount**() Gibt die Anzahl der aktiven Spieler zurück (zur Siegprüfung)
`const CGamefield *`	**GetGamefield** (WORD goid) Ruft gameboard−>GetGamefield() auf.
`const void`	**GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into) Liefert eine Liste der Spielfelder
`const CHat *`	**GetHat**(WORD goid) Liefert der GOID entsprechenden Hut zurück.
`const void`	**GetHats**(CTypedPtrList<CPtrList, CHat*> & into) Liefert eine Liste von Hüte
`const CPlayer *`	**GetPlayer**(WORD goid) Gibt den Spieler mit dieser GOID zurück.
`const void`	**GetPlayers**(CTypedPtrList<CPtrList, CPlayer*> & into) Liefert eine Liste der Spieler
`WORD`	**NextPlayer**() Setzt den nächsten aktiven Spieler anhand des inkrementieren von active_player (evtl.

Method Summary	
`void`	**SetActivePlayer** (WORD goid) Setzt den Spieler mit der erhaltenen ID als aktiv.
`void`	**SetSelectableGamefields** (const CWordArray & fields, BOOL set) Setzt Spielfelder anklickbar
`int`	**ThrowDice**() Ruft dice.ThrowDice() auf.

Constructor Detail

CFDH_Doc

public **CFDH_Doc**()

Erstellt ein Spielfeld und einen Würfel. Die Spieler werden erst
später über AddPlayer() hinzugefügt.

Method Detail

~CFDH_Doc

public synchronized **~CFDH_Doc**()

Method Detail

AddPlayer

public void **AddPlayer**(CString & name, enum playercolor color)

Registriert einen neuen Spieler. Ausserdem werden seine Hüte erstellt und mit ihm verknüpft.

GetActivePlayerCount

public int **GetActivePlayerCount** ()

Gibt die Anzahl der aktiven Spieler zurück (zur Siegprüfung)

GetGamefield

public const CGamefield * **GetGamefield** (WORD goid)

Ruft gameboard−>GetGamefield() auf.

GetGamefields

public const void **GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into)

Liefert eine Liste der Spielfelder

GetHat

public const CHat * **GetHat**(WORD goid)

Liefert der GOID entsprechenden Hut zurück.

GetHats

public const void **GetHats**(CTypedPtrList<CPtrList, CHat*> & into)

Liefert eine Liste von Hüte

GetPlayer

public const CPlayer * **GetPlayer**(WORD goid)

Gibt den Spieler mit dieser GOID zurück.

GetPlayers

public const void **GetPlayers**(CTypedPtrList<CPtrList, CPlayer*> & into)

Liefert eine Liste der Spieler

NextPlayer

public WORD **NextPlayer**()

Setzt den nächsten aktiven Spieler anhand des inkrementieren von active_player (evtl. Überlauf beachten und dann wieder vorne anfangen). Ist dieser Spieler nicht mehr aktiv (!player−>GetState()), beim nächsten probieren. Der Controller muss überprüfen, ob dieser neue aktive Spieler wieder derselbe ist. Ist dies der Fall ist das Spiel gewonnen, da nur noch ein Spieler aktiv ist. Liefert die GOID dieses Spielers zurück.

SetActivePlayer

public void **SetActivePlayer** (WORD goid)

Setzt den Spieler mit der erhaltenen ID als aktiv.

SetSelectableGamefields

public void **SetSelectableGamefields** (const CWordArray & fields, BOOL set)

Setzt Spielfelder anklickbar

ThrowDice

public int **ThrowDice**()

Ruft dice.ThrowDice() auf.

Association Links

to Class CDice

Eine Instanz des Würfels.

Supplier Cardinality 1

to Class CPlayer

Ein CTypedPtrMap aller Spieler, mit GOID als Schlüssel.

Supplier Cardinality 2..4
Type aggregation
associates CPlayer

to Class CGameboard

Eine Instanz des Spielbretts.

Supplier Cardinality 1

to Class CHat

Eine CTypedPtrMap aller Hüte, mit GOID als Schlüssel.

Supplier Cardinality 12..24
associates CHat

Class CFDH_Interface

class CFDH_Interface derived from CDocument

Dies ist das zentrale Interface des Spiels. Es dient der Kommunikation zwischen den "großen Objekten" View/Controller/Document. Ausserdem werden diese hier verwaltet.

Destructor Summary	
virtual synchronized	**~CFDH_Interface**()

Method Summary	
void	**AddPlayer**(CString & name, enum playercolor color) Ruft document−>AddPlayer() und view−>AddPlayer() auf
	DECLARE_DYNCREATE()
int	**GetActivePlayerCount** () ruft document−>GetActivePlayerCount() auf
const CFDH_Controller *	**GetController** () gibt den Pointer auf den Controller zurück.
CGamefield *	**GetGamefield** (WORD goid) ruft document−>GetGamefield() auf.
const void	**GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into) ruft document−>GetGamefield() auf
CHat *	**GetHat**(WORD goid) ruft document−>GetHat() auf.
const void	**GetHats**(CTypedPtrList<CPtrList, CHat*> & into) ruft document−>GetHats() auf
CPlayer *	**GetPlayer** (WORD goid) ruft document−>GetPlayer() auf
const void	**GetPlayers** (CTypedPtrList<CPtrList, CPlayer*> & into) ruft document−>GetPlayers() auf
BOOL	**HatClicked** (WORD goid) ruft controller−>HatClicked() auf.

Method Summary	
`WORD`	**NextPlayer** () Ruft document−>NextPlayer() und view−>SetPlayer() auf.
`virtual BOOL`	**OnNewDocument** ()
`BOOL`	**RButtonPressed** () ruft controller−>RButtonPressed() auf.
`void`	**RegisterView** (CFDH_View * pnew_view)
`virtual void`	**Serialize** (CArchive & ar)
`void`	**SetActiveHat** (WORD goid) ruft view−>SetActiveHat() auf
`void`	**SetActivePlayer** (WORD goid) ruft ctrl−>SetActivePlayer() und view−>SetPlayer() auf
`void`	**SetGamerules** (enum gamerules rules) ruft controller−>SetGamerules() auf
`void`	**SetSelectableGamefields** (const CWordArray & fields, BOOL set) ruft doc−>SetSelectableGamefields() & view−>SetSelectableGamefields() auf.
`void`	**SetThrownNumber** (int number) ruft ctrl−>SetThrownNumer() und view−>SetDice() auf
`void`	**StartGame** () ruft controller−>StartGame() auf
`int`	**ThrowDice** () ruft document−>ThrowDice() und view−>SetDice() auf.
`void`	**ViewSetRegistered** (BOOL value) ruft view−>SetRegistered() auf

Method Detail

~CFDH_Interface

public virtual synchronized **~CFDH_Interface**()

Method Detail

AddPlayer

public void **AddPlayer**(CString & name, enum playercolor color)

Ruft document−>AddPlayer() und view−>AddPlayer() auf

DECLARE_DYNCREATE

public **DECLARE_DYNCREATE**()

GetActivePlayerCount

public int **GetActivePlayerCount** ()

ruft document−>GetActivePlayerCount() auf

GetController

public const CFDH_Controller * **GetController** ()

gibt den Pointer auf den Controller zurück.

GetGamefield

public CGamefield * **GetGamefield** (WORD goid)

ruft document−>GetGamefield() auf.

GetGamefields

public const void **GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into)

ruft document−>GetGamefield() auf

GetHat

public CHat * **GetHat**(WORD goid)

ruft document−>GetHat() auf.

GetHats

public const void **GetHats**(CTypedPtrList<CPtrList, CHat*> & into)

ruft document−>GetHats() auf

GetPlayer

public CPlayer * **GetPlayer**(WORD goid)

ruft document−>GetPlayer() auf

GetPlayers

public const void **GetPlayers**(CTypedPtrList<CPtrList, CPlayer*> & into)

ruft document−>GetPlayers() auf

HatClicked

public BOOL **HatClicked**(WORD goid)

ruft controller−>HatClicked() auf.

NextPlayer

public WORD **NextPlayer**()

Ruft document−>NextPlayer() und view−>SetPlayer() auf.

OnNewDocument

public virtual BOOL **OnNewDocument**()

RButtonPressed

public BOOL **RButtonPressed**()

ruft controller−>RButtonPressed() auf.

RegisterView

public void **RegisterView** (CFDH_View * pnew_view)

Serialize

public virtual void **Serialize** (CArchive & ar)

SetActiveHat

public void **SetActiveHat** (WORD goid)

ruft view−>SetActiveHat() auf

SetActivePlayer

public void **SetActivePlayer** (WORD goid)

ruft ctrl−>SetActivePlayer() und view−>SetPlayer() auf

SetGamerules

public void **SetGamerules** (enum gamerules rules)

ruft controller−>SetGamerules() auf

SetSelectableGamefields

public void **SetSelectableGamefields** (const CWordArray & fields, BOOL set)

ruft doc−>SetSelectableGamefields() & view−>SetSelectableGamefields() auf.

SetThrownNumber

public void **SetThrownNumber** (int number)

ruft ctrl−>SetThrownNumer() und view−>SetDice() auf

StartGame

public void **StartGame**()

ruft controller−>StartGame() auf

ThrowDice

public int **ThrowDice**()

ruft document−>ThrowDice() und view−>SetDice() auf.

ViewSetRegistered

public void **ViewSetRegistered** (BOOL value)

ruft view−>SetRegistered() auf

Association Links

to Class CFDH_Doc

Zeiger auf das Dokument mit allen Spieldaten

to Class CFDH_View

Zeiger auf View

to Class CFDH_Controller

Zeiger auf Controller (Spiellogik)

to Class CSaveState

Zeiger auf Spielstandspeicher

Class CFDH_View

class CFDH_View derived from CView

View enthält Funktionalität zur Darstellung des Spielbrettes, der Spielfiguren, des Würfels und der Spielerinformationen enthalten. Die Klasse besitzt keine Anwendungslogik und ist nur für die Darstellung zuständig.

Destructor Summary	
virtual synchronized	**~CFDH_View**()

Method Summary	
void	**AddPlayer**(CString & name, enum playercolor color) Legt CSimpleHat Objekte an für diesen Spieler.
	DECLARE_DYNCREATE()
CFDH_Interface *	**GetDocument** () Überladung von GetDocument() die den Pointer auf das Interface zurückgibt
const CDrawableHat *	**GetDrawableHat**(WORD goid) Liefert das zur GOID passende DrawableHatObject zurück um dessen Position setzen zu können.
CGamefield *	**GetGamefield** (WORD goid) Liefert das mit dieser GOID identifizierte Spielfeld zurück.
CHat *	**GetHat**(WORD goid) Liefert den Hut mit dieser GOID zurück.
virtual void	**OnDraw**(CDC * pDC)
void	**ResetGame** () Setzt den View so zurück, dass ein neues Spiel dargestellt werden kann
void	**SetActiveHat** (WORD goid) Setzt den selektierten Hut im Preview
void	**SetDice** (int number) ruft dice.SetValue() auf.

Method Summary

void	**SetPlayer** (WORD goid) Setzt den Spielernamen des aktuell aktiven Spielers auf diese GOID bzw. wird über diese mittels GetPlayer() auf die eigentlichen Player−Daten zugegriffen und dem playerdisplay per SetString() und SetColor() übermittelt.
void	**SetRegistered** (BOOL value) Setzt die Variable registered
void	**SetSelectableGamefields** (const CWordArray & fields, BOOL set) Setzt alle angegebenen GOIDs als selektierbar oder nicht selektierbar.

Method Detail

~CFDH_View

public virtual synchronized **~CFDH_View**()

Method Detail

AddPlayer

public void **AddPlayer**(CString & name, enum playercolor color)

Legt CSimpleHat Objekte an für diesen Spieler.

DECLARE_DYNCREATE

public **DECLARE_DYNCREATE**()

GetDocument

public CFDH_Interface * **GetDocument** ()

Überladung von GetDocument() die den Pointer auf das Interface zurückgibt

---

GetDrawableHat

public const CDrawableHat * **GetDrawableHat**(WORD goid)

Liefert das zur GOID passende DrawableHatObject zurück um dessen Position setzen zu können.

GetGamefield

public CGamefield * **GetGamefield** (WORD goid)

Liefert das mit dieser GOID identifizierte Spielfeld zurück.

GetHat

public CHat * **GetHat**(WORD goid)

Liefert den Hut mit dieser GOID zurück.

OnDraw

public virtual void **OnDraw**(CDC * pDC)

ResetGame

public void **ResetGame**()

Setzt den View so zurück, dass ein neues Spiel dargestellt werden kann

---

SetActiveHat

public void **SetActiveHat**(WORD goid)

Setzt den selektierten Hut im Preview

SetDice

public void **SetDice** (int number)

ruft dice.SetValue() auf.

SetPlayer

public void **SetPlayer** (WORD goid)

Setzt den Spielernamen des aktuell aktiven Spielers auf diese GOID bzw. wird über diese mittels GetPlayer() auf die eigentlichen Player−Daten zugegriffen und dem playerdisplay per SetString() und

SetColor() übermittelt.

SetRegistered

public void **SetRegistered** (BOOL value)

Setzt die Variable registered

SetSelectableGamefields

public void **SetSelectableGamefields** (const CWordArray & fields, BOOL set)

Setzt alle angegebenen GOIDs als selektierbar oder nicht selektierbar.

Association Links

to Class CDrawableDice

Der darzustellende Würfel

Supplier Cardinality 1

to Class CDrawableHatPreview

Der Preview des Hutes + seiner Gefangenen

Supplier Cardinality 1

to Class CDrawableTextlabel

Ein buntes Label mit dem Namen des aktiven Spielers

Supplier Cardinality 1

to Class CDrawableTextlabel

Zeiger auf den aktuellen Spieler

Class CGameboard

---

class CGameboard

Das Spielbrett ist eigentlich nichts anderes als ein Container für die Spielfelder.

Constructor Summary

CGameboard()
Hier bekommen alle Spielfelder ihre Eigenschaften und werden miteinander verknüpft.

Destructor Summary

~CGameboard()
Löscht alle angelegten Spielfeld−Objekte.

Method Summary

`const CGamefield *`	**GetGamefield** (WORD goid) Gibt das Spielfeld mit der entsprechenden GameObjectID zurück.
`const void`	**GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into)

Constructor Detail

CGameboard

public **CGameboard**()

Hier bekommen alle Spielfelder ihre Eigenschaften und werden miteinander verknüpft.

Method Detail

~CGameboard

public synchronized **~CGameboard**()

Löscht alle angelegten Spielfeld−Objekte.

Method Detail

GetGamefield

public const CGamefield * **GetGamefield** (WORD goid)

Gibt das Spielfeld mit der entsprechenden GameObjectID zurück.

GetGamefields

public const void **GetGamefields** (CTypedPtrList<CPtrList, CGamefield*> & into)

Association Links

to Class CGamefield

Hier werden alle 81 Spielfelder in einer CTypedPtrMap festgehalten. Der Schüssel für diese Map ist die GameObjectID des Spielfeldes.

Supplier Cardinality 81
Type aggregation
associates CGamefield

Class CGamefield

class CGamefield

Datenobjekt Spielfeld, mit Verknüpfung auf Spielfeldnachbarn und Verknüpfung mit allen Hüten die auf dem Feld stehen.

Constructor Summary
CGamefield ()

Destructor Summary
~CGamefield ()

Method Summary	
`void`	**AddHat**(CHat * hat) Fügt einen Hut in die Liste der Hüte an.
`const WORD`	**GetGOID**() Liefert die GOID dieses Feldes zurück.
`const void`	**GetHats**(CTypedPtrList<CPtrList, CHat*> & into) Gibt eine Listen mit allen sich auf dem Feld befindlichen Hüten zurück.
`const void`	**GetNeighbours** (CWordArray & into, WORD without_field) Diese Funktion wird aufgerufen wenn berechnet wird, auf welche Felder man laufen kann.
`const BOOL`	**GetSelectable** () Gibt zurück ob dieses Feld auswählbar ist.
`bool`	**RemoveHat**(CHat * hat) Löscht den angegebenen Hut aus der Liste.
`void`	**SetGOID** (WORD new_goid) Setzt die GOID des Spielfeldes.
`void`	**SetHats** (const CTypedPtrList<CPtrList, CHat*> & src) Setzt die Liste an auf diesem Feld befindlichen Hüte

Method Summary	
`void`	**SetNeighbours** (const CWordArray & fields) Setzt die Liste an Nachbarn.
`void`	**SetSelectable** (BOOL new_value) Setzt ob diese Feld selektierbar ist.

Constructor Detail

CGamefield

public **CGamefield** ()

Method Detail

~CGamefield

public synchronized **~CGamefield** ()

Method Detail

AddHat

public void **AddHat**(CHat * hat)

Fügt einen Hut in die Liste der Hüte an.

GetGOID

public const WORD **GetGOID**()

Liefert die GOID dieses Feldes zurück.

GetHats

public const void **GetHats**(CTypedPtrList<CPtrList, CHat*> & into)

Gibt eine Listen mit allen sich auf dem Feld befindlichen Hüten zurück.

GetNeighbours

public const void **GetNeighbours** (CWordArray & into, WORD without_field)

Diese Funktion wird aufgerufen wenn berechnet wird, auf welche Felder man laufen kann. Sie gibt eine Liste an Spielfeld−Nachbarn zurück, allerdings ohne das Feld das per Parameter übermittelt wurde (verhindert Überschneidungen)

GetSelectable

public const BOOL **GetSelectable** ()

Gibt zurück ob dieses Feld auswählbar ist.

RemoveHat

public bool **RemoveHat**(CHat * hat)

Löscht den angegebenen Hut aus der Liste. Gibt true zurück bei Erfolg, false falls der Hut nicht in der Liste existierte.

SetGOID

public void **SetGOID** (WORD new_goid)

Setzt die GOID des Spielfeldes.

SetHats

public void **SetHats** (const CTypedPtrList<CPtrList, CHat*> & src)

Setzt die Liste an auf diesem Feld befindlichen Hüte

SetNeighbours

public void **SetNeighbours** (const CWordArray & fields)

Setzt die Liste an Nachbarn.

SetSelectable

public void **SetSelectable** (BOOL new_value)

Setzt ob diese Feld selektierbar ist.

Association Links

to Class CHat

Alle sich auf diesem Spielfeld befindlichen Hüte in einer Liste

Supplier Cardinality 0..*
associates CHat

Class CHat

class CHat

Repräsentiert ein Spielobjekt "Hut" mit seinem Status, Besitzer, besetzes Feld und die Hüte die dieser Hut gefangen hat.

Constructor Summary

CHat()

Destructor Summary

~CHat()

Method Summary

`void`	**Catch**(CHat * hat) Nimmt den angegebenen Hut gefangen.
`void`	**ClearList**() Löscht die Liste der Gefangenen.
`const void`	**GetCaughtHats**(CTypedPtrList<CPtrList, CHat*> & into) Gibt die Liste aller Gefangenen als Liste zurück.
`const CGamefield *`	**GetGamefield** () Liefert das Feld zurück auf dem dieser Hut steht.
`const WORD`	**GetGOID**() Liefert die GOID dieses Hutes zurück.
`const CPlayer *`	**GetOwner**() Gibt den Besitzer dieses Hutes zurück.
`void`	**SetCaughtHats** (const CTypedPtrList<CPtrList, CHat*> & src) Setzt die Liste der Gefangenen
`void`	**SetGamefield** (CGamefield * new_gamefield) Setzt das Feld auf dem dieser Hut steht.
`void`	**SetGOID** (WORD new_goid) Setzt die GOID des Hutes.

Method Summary	
void	**SetOwner** (CPlayer * new_owner) Setzt den Besitzer dieses Hutes.
void	**SetState** (enum hatstate new_state) Setzt den Status des Hutes.

Constructor Detail

CHat

public **CHat**()

Method Detail

~CHat

public synchronized **~CHat**()

Method Detail

Catch

public void **Catch**(CHat * hat)

Nimmt den angegebenen Hut gefangen. Holt sich seine Gefangenliste und hängt die an sich dran. Löscht die Liste der Gefangenen des Gefangenen.

ClearList

public void **ClearList**()

Löscht die Liste der Gefangenen.

GetCaughtHats

public const void **GetCaughtHats**(CTypedPtrList<CPtrList, CHat*> & into)

Gibt die Liste aller Gefangenen als Liste zurück.

GetGamefield

public const CGamefield * **GetGamefield** ()

Liefert das Feld zurück auf dem dieser Hut steht.

GetGOID

public const WORD **GetGOID**()

Liefert die GOID dieses Hutes zurück.

GetOwner

public const CPlayer * **GetOwner**()

Gibt den Besitzer dieses Hutes zurück.

SetCaughtHats

public void **SetCaughtHats** (const CTypedPtrList<CPtrList, CHat*> & src)

Setzt die Liste der Gefangenen

SetGamefield

public void **SetGamefield** (CGamefield * new_gamefield)

Setzt das Feld auf dem dieser Hut steht.

SetGOID

public void **SetGOID** (WORD new_goid)

Setzt die GOID des Hutes.

SetOwner

public void **SetOwner** (CPlayer * new_owner)

Setzt den Besitzer dieses Hutes.

SetState

public void **SetState** (enum hatstate new_state)

Setzt den Status des Hutes.

Association Links

to Class CPlayer

Zeigt auf den Besitzer des Hutes, vereinfacht Zugriff auf die anderen Hüte desselben Besitzers.

Supplier Cardinality 1

to Class CGamefield

Zeigt auf das Feld auf welchem dieser Hut sichgerade befindet.

Supplier Cardinality 1

Class CNewGameDialog

---

class CNewGameDialogderived from CDialog

Maske zum Starten eines neuen Spiels. Dient der Spieleranmeldung.

Constructor Summary
CNewGameDialog (CWnd * pParent)

Method Summary	
`void`	**RegisterInterface** (CFDH_Interface * pnewiface)

Constructor Detail

CNewGameDialog

public **CNewGameDialog** (CWnd * pParent)

Method Detail

RegisterInterface

public void **RegisterInterface** (CFDH_Interface * pnewiface)

Association Links

to Class CFDH_Interface

Class CPlayer

---

class CPlayer

Repräsentiert einen Spieler in Fang den Hut.

Constructor Summary
CPlayer()

Destructor Summary
~CPlayer()

Method Summary	
`const WORD`	**GetGOID**() Liefert die GOID dieses Spielers zurück.
`const void`	**GetHats**(CTypedPtrList<CPtrList, CHat*> & into) Gibt eine CTypedPtrList zurück mit allen Hüten dieses Spielers.
`const CString`	**GetName**() Gibt den Namen dieses Spielers zurück.
`const BOOL`	**GetState** () Diese Funktion schaut für alle Hüte nach, ob sie noch frei sind.
`void`	**SetGOID** (WORD new_goid) Setzt die GOID des Spielers.
`void`	**SetHats** (CTypedPtrList<CPtrList, CHat*> & new_hats) Setzt die Liste an Hüten für diesen Spieler
`void`	**SetName** (CString & new_name) Setzt den Namen des Spielers.

Constructor Detail

CPlayer

public **CPlayer**()

52, CPlayer

Method Detail

~CPlayer

public synchronized **~CPlayer**()

Method Detail

GetGOID

public const WORD **GetGOID**()

Liefert die GOID dieses Spielers zurück.

GetHats

public const void **GetHats**(CTypedPtrList<CPtrList, CHat*> & into)

Gibt eine CTypedPtrList zurück mit allen Hüten dieses Spielers.

GetName

public const CString **GetName**()

Gibt den Namen dieses Spielers zurück.

GetState

public const BOOL **GetState**()

Diese Funktion schaut für alle Hüte nach, ob sie noch frei sind. Ist dies der Fall wird true (Spieler noch aktiv) zurückgegeben, ansonsten false (Spieler besiegt)

SetGOID

public void **SetGOID**(WORD new_goid)

Setzt die GOID des Spielers.

SetHats

public void **SetHats** (CTypedPtrList<CPtrList, CHat*> & new_hats)

Setzt die Liste an Hüten für diesen Spieler

SetName

public void **SetName** (CString & new_name)

Setzt den Namen des Spielers.

Association Links

to Class CHat

Die Hüte dieses Spielers. Immer 6. Sie werden in einer Liste gehalten deren Schlüssel die GOID des Hutes ist.

Supplier Cardinality 6
associates CHat

Class CSaveState

class CSaveState

CSaveState ist für die Speicherung und Wiederherstellung der Spielzustände zuständig

Constructor Summary
CSaveState ()

Destructor Summary
~CSaveState ()

Method Summary	
`void`	**RegisterInterface** (CFDH_Interface * pnewiface)
`void`	**RestoreStateFromArchive** (CArchive & ar) Stellt einen Zustand wieder her.
`void`	**SaveStateToArchive** (CArchive & ar) Speichert den aktuellen Zustand.

Constructor Detail

CSaveState

public **CSaveState** ()

Method Detail

~CSaveState

public synchronized **~CSaveState** ()

Method Detail

RegisterInterface

public void **RegisterInterface** (CFDH_Interface * pnewiface)

RestoreStateFromArchive

public void **RestoreStateFromArchive** (CArchive & ar)

Stellt einen Zustand wieder her.

SaveStateToArchive

public void **SaveStateToArchive** (CArchive & ar)

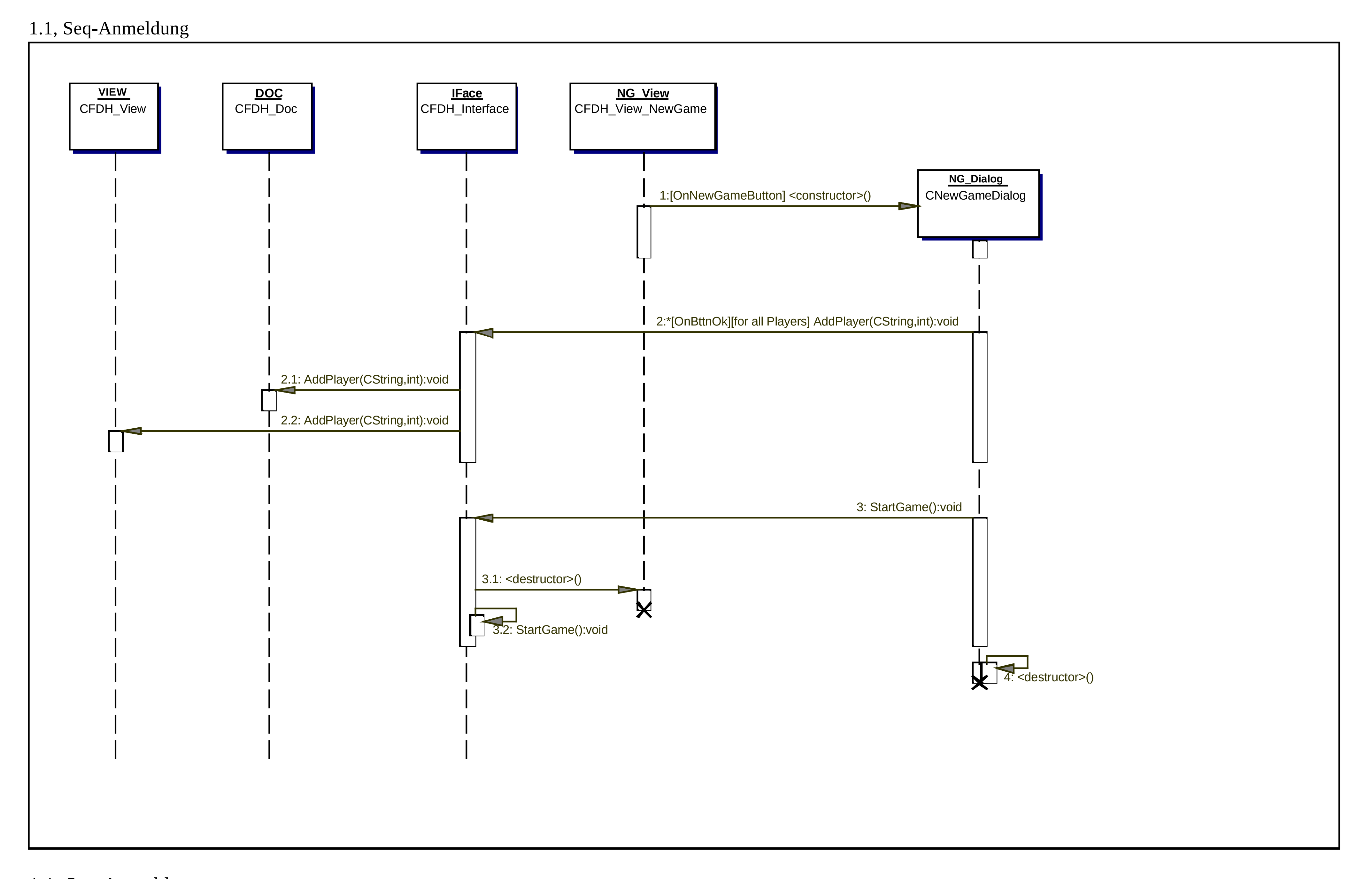
Speichert den aktuellen Zustand.

Association Links

to Class CFDH_Interface

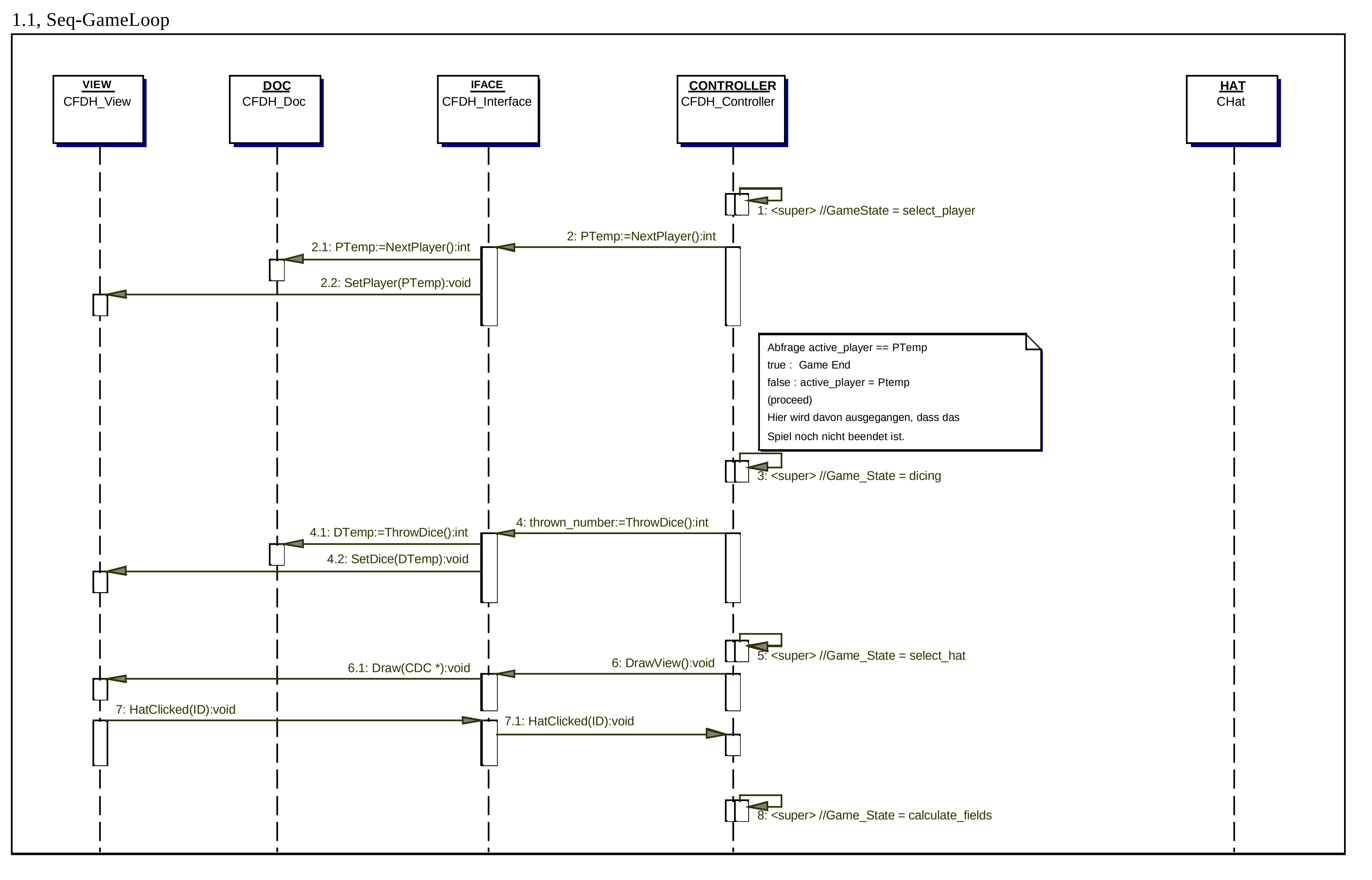
D. Sequenzdiagramm – Anmeldung

1.1, Seq-Anmeldung

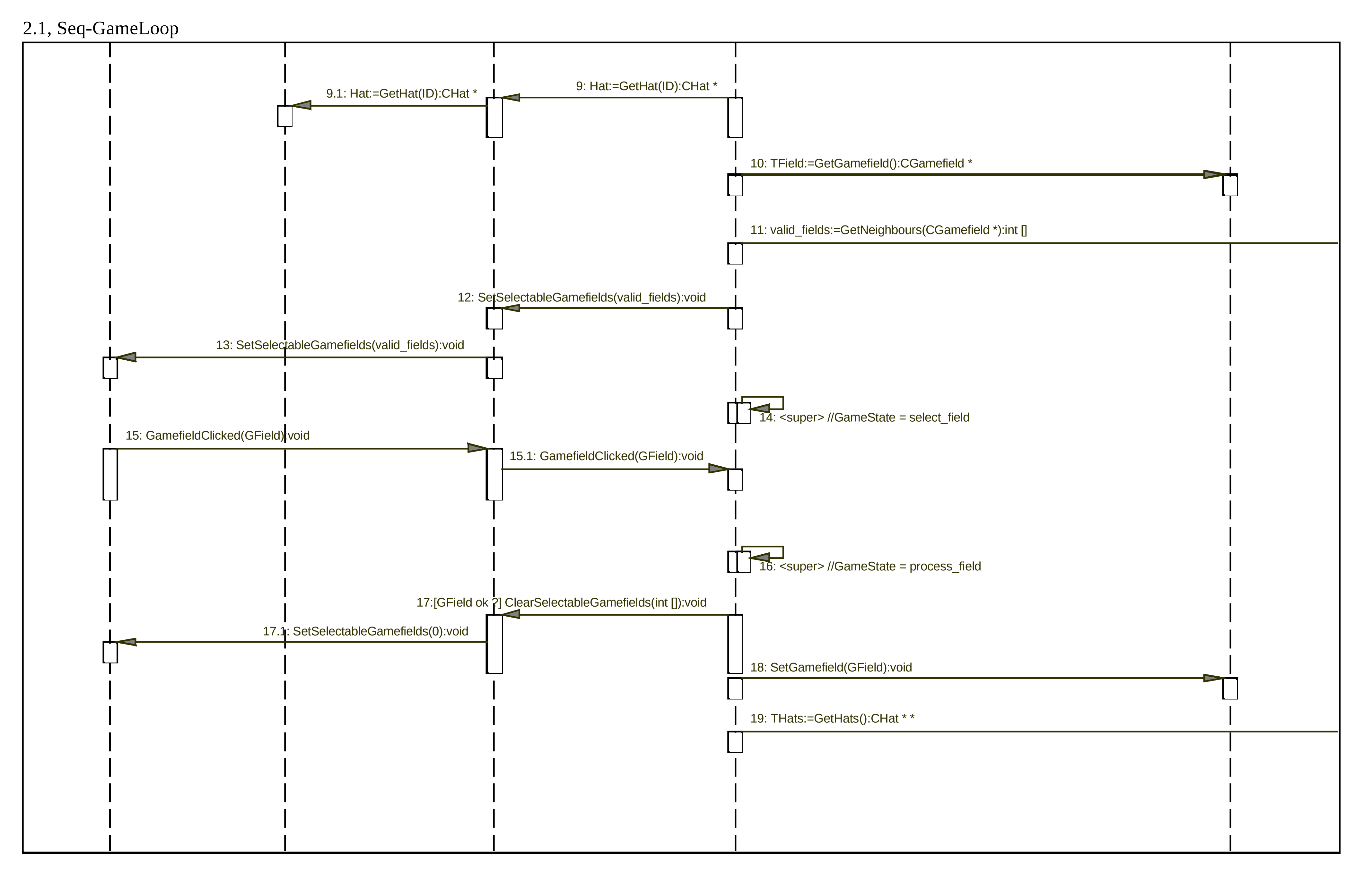


1.1, Seq-Anmeldung

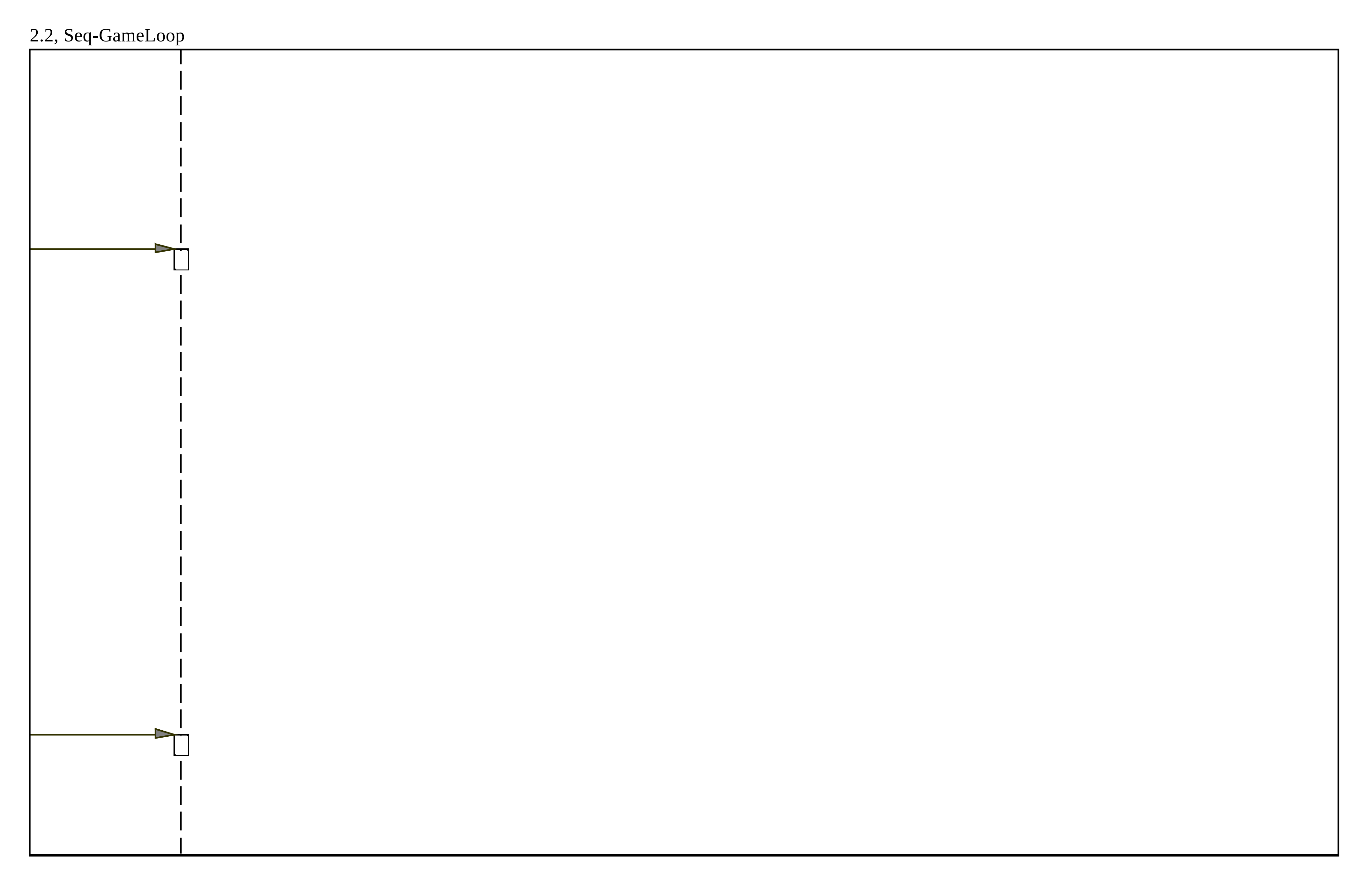
E. Sequenzdiagramm – GameLoop


VIEW
CFDH_View
DOC
CFDH_Doc
IFACE
CFDH_Interface
CONTROLLER
CFDH_Controller
HAT
CHat
1: <super> //GameState = select_player
2: PTemp:=NextPlayer():int
2.1: PTemp:=NextPlayer():int
2.2: SetPlayer(PTemp):void
Abfrage active_player == PTemp
true : Game End
false : active_player = Ptemp
(proceed)
Hier wird davon ausgegangen, dass das
Spiel noch nicht beendet ist.
3: <super> //Game_State = dicing
4: thrown_number:=ThrowDice():int
4.1: DTemp:=ThrowDice():int
4.2: SetDice(DTemp):void
5: <super> //Game_State = select_hat
6: DrawView():void
6.1: Draw(CDC *):void
7: HatClicked(ID):void
7.1: HatClicked(ID):void
8: <super> //Game_State = calculate_fields

FIELD
CGamefield


9: Hat:=GetHat(ID):CHat *
9.1: Hat:=GetHat(ID):CHat *
10: TField:=GetGamefield():CGamefield *
11: valid_fields:=GetNeighbours(CGamefield *):int []
12: SetSelectableGamefields(valid_fields):void
13: SetSelectableGamefields(valid_fields):void
14: <super> //GameState = select_field
15: GamefieldClicked(GField):void
15.1: GamefieldClicked(GField):void
16: <super> //GameState = process_field
17:[GField ok ?] ClearSelectableGamefields(int []):void
17.1: SetSelectableGamefields(0):void
18: SetGamefield(GField):void
19: THats:=GetHats():CHat * *

2.2, Seq-GameLoop



2.2, Seq-GameLoop

Die Folgende Iteration wird nur
durchlaufen,wenn der Zug auf einem
Normalfeld beendet
wurde.Andernfalls springt das Programm
direkt ans Ende.

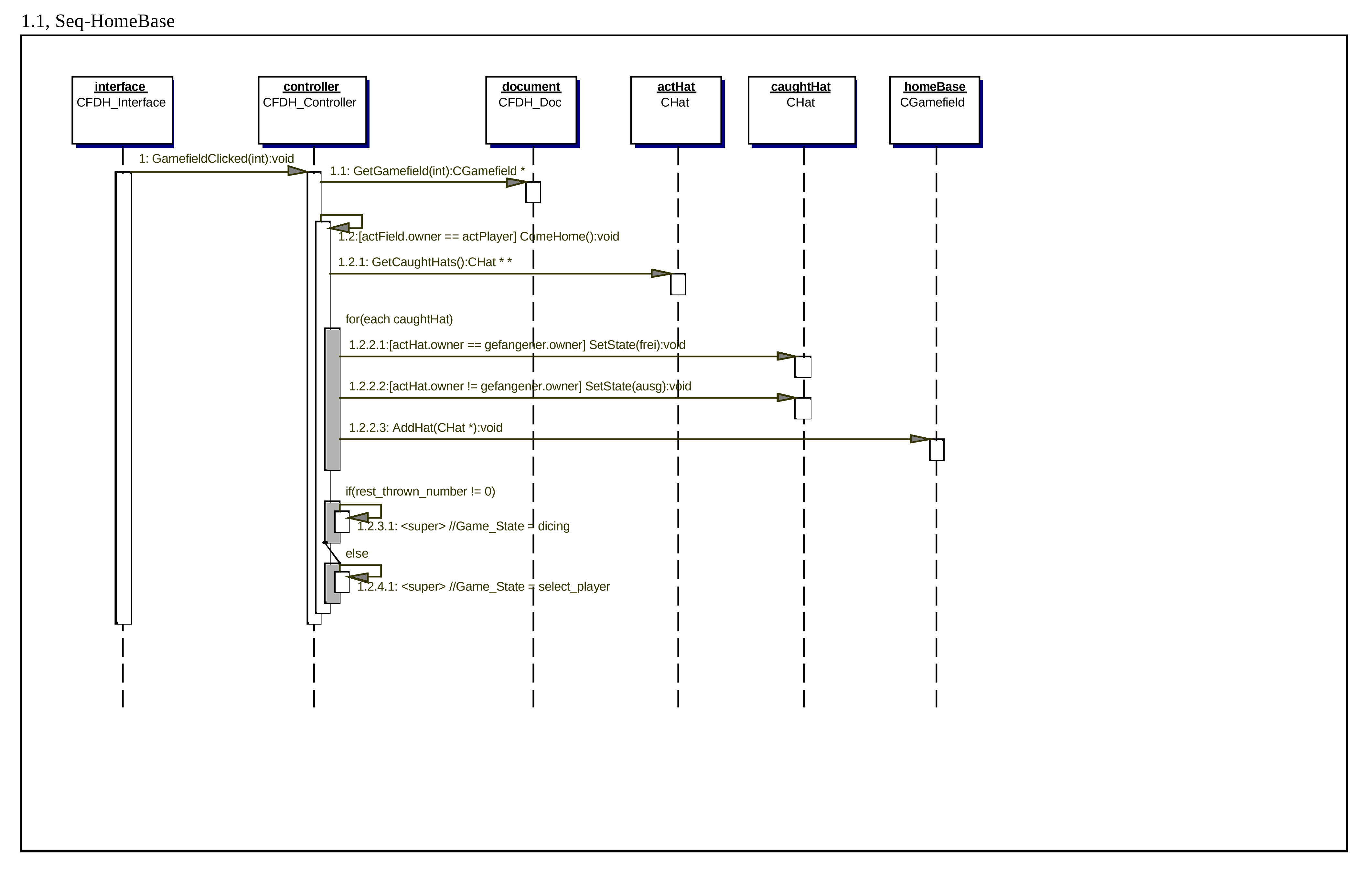
20:*[THats][if THat-Owner != Hat-Owner] Catch(THat):void

Hier beginnt die Schleife von
Neuem.

3.2, Seq-GameLoop

3.2, Seq-GameLoop

F. Sequenzdiagramm – HomeBase


interface
CFDH_Interface
controller
CFDH_Controller
document
CFDH_Doc
actHat
CHat
caughtHat
CHat
homeBase
CGamefield
1: GamefieldClicked(int):void
1.1: GetGamefield(int):CGamefield *
1.2:[actField.owner == actPlayer] ComeHome():void
1.2.1: GetCaughtHats():CHat * *
for(each caughtHat)
1.2.2.1:[actHat.owner == gefangener.owner] SetState(frei):void
1.2.2.2:[actHat.owner != gefangener.owner] SetState(ausg):void
1.2.2.3: AddHat(CHat *):void
if(rest_thrown_number != 0)
1.2.3.1: <super> //Game_State = dicing
else
1.2.4.1: <super> //Game_State = select_player